



Programación Orientada a Objetos

Clase 1 - Generalidades – Jerarquía de clases

UTN - La Plata



Introducción

1. El proceso de abstracción.
2. Objetos. Estado y comportamiento. Composición.
3. Clases e Instancias.
4. Mensajes y métodos. Invocación de métodos.
5. Encapsulamiento y ocultamiento de información.
6. Herencia. Polimorfismo. Binding dinámico.
7. Actividad 1.

1. El proceso de abstracción

Las aplicaciones de software modelan el mundo real.



Los humanos construimos modelos mentales para entender el mundo.



Un modelo mental es una visión simplificada de cómo las cosas funcionan y cómo podemos interactuar con ellas.

El mundo real es complejo, la abstracción es uno de los mecanismos para combatir la complejidad.

Los paradigmas de programación proveen un conjunto de herramientas conceptuales que nos permiten analizar, representar y abordar los problemas.

1. El proceso de abstracción



Cada **paradigma de programación** determina la manera de pasar del espacio de los problemas al de la implementación de una solución.



Lenguajes pertenecientes a los diferentes paradigmas de programación:

Programación Imperativa: Ej. Pascal, C, Ada

Programación Orientada a objetos: Ej. Smalltalk, Java, Python, C++

Programación Funcional: Ej. Lisp, Haskell

Programación Lógica: Ej. Prolog

1. El proceso de abstracción

El objetivo de la Programación Orientada a objetos es manejar la complejidad de los problemas del mundo real, abstrayendo su conocimiento y encapsulándolo en objetos.

La Programación Orientada a Objetos describe un sistema en términos de los objetos involucrados. Dichos objetos interactúan entre sí enviándose mensajes para llevar a cabo una tarea.

Cuando debemos realizar un sistema lo primero que debemos hacer es identificar los objetos involucrados en el sistema.

Es importante quedarnos solamente con aquellos objetos que son de nuestro interés, teniendo siempre presente que es lo que queremos modelar.

Identificar los objetos es un arte y no una ciencia.

El resultado dependerá del contexto y punto de vista de quien modela.

2. Objetos. Estado y comportamiento. Composición

Cada **objeto** abstrae un dato del programa y lo que puede hacerse sobre él.

¿Qué es un objeto?

Objeto: es una entidad que tiene dos características: ***estado y comportamiento***.

Estado: está representado por los ***atributos***, es decir las propiedades relevantes de un objeto. Los atributos de un objeto se almacenan en ***variables de instancia***.

Comportamiento: está representado por una ***serie de funciones o métodos*** que modifican o no el estado del objeto.

A la representación de un objeto de la vida real en un programa se lo denomina **objeto de software**.

2. Objetos. Estado y comportamiento. Composición

Ejemplos:

Una **perro** tiene
estado: nombre, raza, color
comportamiento: ladrar, jugar, comer, etc.



Una **bicicleta** tiene
estado: cadena, nº de cambios, cambio actual
comportamiento: acelerar, frenar, etc.



También podemos representar **objetos abstractos**, es decir objetos que no tienen una representación física en el mundo real.

Una **Caja de ahorro**



1 208 000,00	1 150 500,00
34 250 000,00	35 300 000,00
84 1 800,00	788 500,00
502 100,00	474 500,00
158 200,00	261 900,00
25 081 000,00	23 471 000,00
62 088 800,00	61 447 500,00

Una **Cuenta corriente**

Composición de objetos

Un objeto puede componerse de dos o más objetos, conformando así un **objeto compuesto**.



Un curso está formado por alumnos



3. Clases e Instancias

Una clase es un “modelo” para definir objetos que pueden realizar las mismas acciones y poseen características similares.

Todas las personas comparten los mismos atributos: nombre, color de pelo, estatura, peso, color de ojos y tienen más o menos un mismo comportamiento como caminar, saltar, correr, comer, etc.

Por lo tanto creamos la **Clase Persona** que agrupa esos atributos y comportamientos comunes.



Estado, vista interna
del objeto



Comportamiento, vista
externa del objeto



Clase Persona

nombre
color de pelo
estatura
peso
color de ojos
caminar
saltar
correr
comer
dormir



3. Clases e Instancias

Una **instancia** es un objeto en particular de una clase.



Roco es una instancia de la clase Perro

Ana, Abril y Belén son instancias de la clase Alumno



Silvia es una instancia de la clase Cliente

El **auto** de Andrés es una instancia de la clase Auto



El **Banco** donde pago mis cuentas es una instancia de la clase Banco

Agustina es una instancia de la clase Empleado



4. Mensajes y métodos. Invocación de métodos

Definición de un problema:

Supongamos que Pablo quiere hacer un pedido de helado para que se lo entreguen en su domicilio.

Pablo, llama a la heladería. Lo atiende la telefonista, Silvina.

Pablo solicita el pedido, indicándole la cantidad, gustos que desea y el domicilio donde debe enviarse el pedido.

Pablo se comunicó con la telefonista, y le envió un mensaje con el requerimiento.

Telefonista,
agente
apropiado

Silvina, la telefonista tiene la
responsabilidad de satisfacer el
requerimiento.



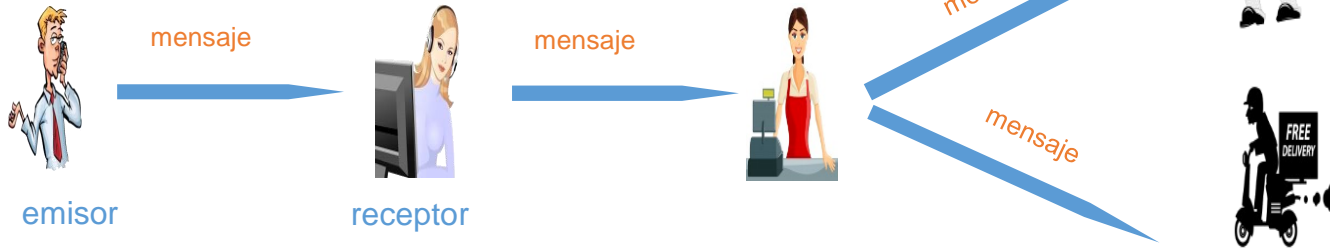
Mensaje con el requerimiento

Pablo no necesita saber cómo
la telefonista resolverá el
problema.

La telefonista usará algún método
para satisfacer el requerimiento.

4. Mensajes y métodos. Invocación de métodos

La resolución del problema, puede requerir la ayuda de otros individuos.



Cada **objeto** cumple un rol, ejecuta una acción, que es usada por otros miembros de la comunidad.

El **emisor** envía un mensaje al **receptor**, junto con los argumentos necesarios para llevar a cabo el requerimiento.

El **receptor** es el objeto a quien se le envía el mensaje.

El receptor en respuesta al mensaje ejecutará un **método** para satisfacer el requerimiento.

Un método es la implementación de un mensaje.

4. Mensajes y métodos. Invocación de métodos

Para que un objeto lleve a cabo un método debe enviársele un MENSAJE. El objeto receptor de dicho mensaje responde ejecutando el METODO asociado. Es decir, el envío de un mensaje a un objeto no es más que la invocación al método correspondiente.

El formato de un mensaje es: `OR mensaje` donde OR es el objeto que recibe el mensaje, al cual se le solicita que ejecute el método correspondiente.

Ej. Sea **alu** un objeto de la clase Alumno y queremos consultarle su nombre.

`res:=alu verNombre.` \leftarrow es equivalente a invocar una función en C \rightarrow `res= verNombre(alu)`

Esto desencadena la ejecución del método solicitado que responde al requerimiento: en este caso retorna el nombre del alumno

5. Encapsulamiento y ocultamiento de información

Encapsulamiento: una clase encapsula la *representación privada* y el *comportamiento de un objeto*.

Un objeto no conoce el funcionamiento interno de los demás objetos y no lo necesita para poder interactuar con ellos.

Ocultamiento de la información: Sólo puede modificarse la estructura interna de un objeto, desde fuera de la clase, a través de la interfaz pública del objeto, es decir sólo a través de los mensajes que entiende el objeto.

Para consultar o modificar el valor almacenado en variables de instancia de un objeto solo puede hacerse mediante los métodos definidos en la clase para tal fin.

6. Herencia. Polimorfismo. Binding dinámico

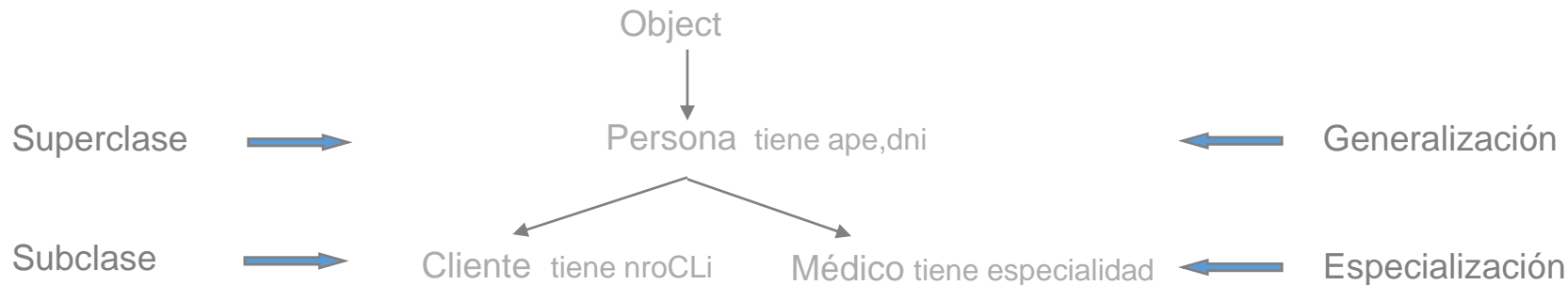
Herencia:

Es la propiedad de crear nuevas clases a partir de otras ya existentes, ampliando su estructura y comportamiento.

Las clases están organizadas en una jerarquía de clases, donde las subclases heredan estado y comportamiento de las superclases.

Las subclases pueden agregar nuevos atributos y comportamiento y además pueden cambiar el comportamiento de los métodos heredados.

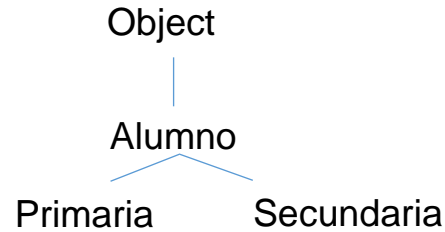
Jerarquía de clases



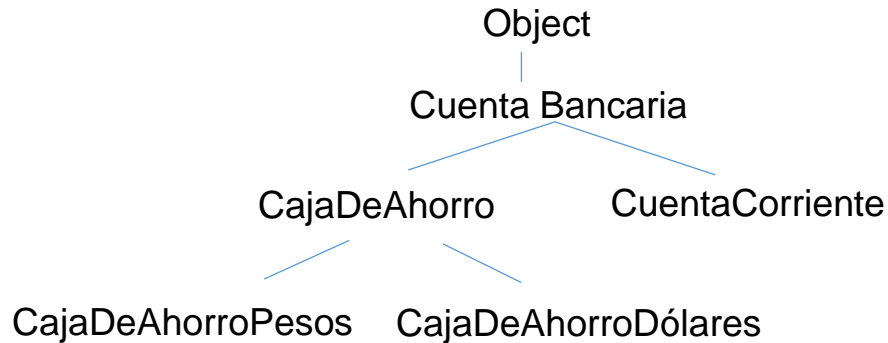
6. Herencia. Polimorfismo. Binding dinámico

Jerarquía de clases

Ejemplo 1:



Ejemplo 2:

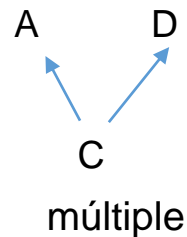
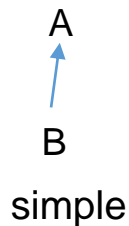


En una jerarquía debe respetarse la relación **es un**.

6. Herencia. Polimorfismo. Binding dinámico

La herencia puede ser:

- De estado: se heredan solo los atributos de la superclase
- De comportamiento: se heredan solo los métodos de la superclase
- Total o parcial: se heredan todos los atributos y métodos(total), o solo parte de ellos
- Simple o múltiple: se hereda de una sola superclase, o se hereda de varias superclases



6. Herencia. Polimorfismo. Binding dinámico

Cuando se programa usando herencia, se programa por extensión:

- Se extiende el código existente, se reusa código. No se vuelve a escribir lo heredado. Sólo se agrega código correspondiente a los nuevos atributos y a las operaciones nuevas.
- Cuando se le envía un mensaje a un objeto, el S.O. lo busca en la clase de dicho objeto. Si no lo encuentra, porque es heredado, sube por el árbol jerárquico de clases aplicando una BÚSQUEDA LOOK-UP, hasta encontrarlo y lo ejecuta.

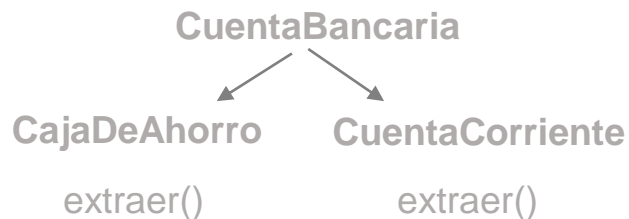
6. Herencia. Polimorfismo. Binding dinámico

Polimorfismo

Es la capacidad que tienen objetos diferentes de entender y responder a un mismo mensaje de manera diferente.

La interpretación de un mensaje es determinado por el receptor y podría variar para diferentes receptores.

Supongamos la siguiente jerarquía:



Ambas clases entienden el mensaje extraer.

El mensaje extraer está implementado de manera diferente en cada clase.

6. Herencia. Polimorfismo. Binding dinámico

Polimorfismo

Supongamos los siguientes ejemplos:

'hola' size.

vec=Array new:5.

vec size.

Ambas clases entienden el mensaje size.

El método size está implementado de manera diferente en cada clase.

3 + 2.

(1/5) + (2/3).

Ambas clases entienden el mensaje +.

El método + está implementado de manera diferente en cada clase.

6. Herencia. Polimorfismo. Binding dinámico

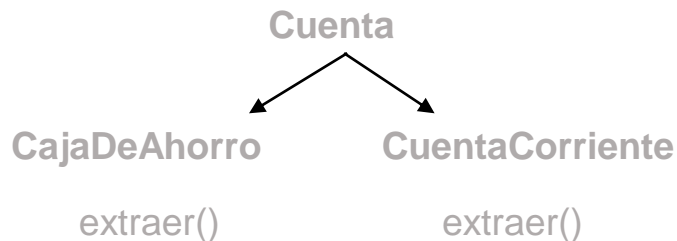
Binding dinámico

Que el binding sea dinámico significa que la ligadura entre el objeto receptor y el método que se va a ejecutar se realizará en tiempo de ejecución.

Ejemplo:

Si el mensaje extraer es enviado a:

una instancia de CajaDeAhorro, va a buscar el mensaje extraer a la clase CajaDeAhorro



una instancia de CuentaCorriente, va a buscar el mensaje extraer a la clase CuentaCorriente

En tiempo de ejecución se va a conocer qué método se va a ejecutar ya que está determinado por el objeto receptor

6. Herencia. Polimorfismo. Binding dinámico

En general, los lenguajes OO no son tipados, es decir, no se declaran las variables. Hay binding dinámico entre la clase del objeto y la variable que apunta a dicho objeto.

|x |

x:=3. → x es de la clase Integer

x class. “muestra Integer”

.....

x:='Perro'. → ahora x es de la clase String

x class. “muestra String”

7. Actividad 1

1) Armar una jerarquía con los siguientes objetos:

a) Bibliotecario LibroPrestarSala Socio Biblioteca LibroPrestarCasa SalaLectura

b) Vivero Planta Empleado Arbol Propietario Arbusto Cliente

