

TALLER DE LENGUAJES II

Cursada 2023

Profesoras: Claudia Queiruga, Jorge Rosso

JTP: Francisco José Blanco

Ayudantes: Jose Arcidiácono, Agustín Cao,
Marcos Moscoso

TALLER DE LENGUAJES II

Días y Horarios

Teoría	Lunes 16:00hs - 18:00hs - Aula 10 A
Trabajos Prácticos	Lunes 18:30hs - 20:30hs - Aula 10 A Jueves 18:30hs - 20:30hs - Aula 8
Plataforma virtual	<u>https://asignaturas.linti.unlp.edu.ar</u>

TALLER DE LENGUAJES II

¿Qué vamos a aprender? ¿Cómo trabajaremos?
¿Qué herramientas usaremos?

Objetivos de aprendizaje

- Comprender el paradigma de programación orientada a objetos, sus características, ventajas y aplicaciones dentro del desarrollo de sistemas de software.
- Desarrollar prácticas concretas y significativas en lenguaje Java.

Metodología de trabajo

A lo largo de la cursada se trabajará:

- sobre **contenidos teóricos** en clases **presenciales** que se articulan con los prácticos.
- en **trabajos prácticos** en los que se pondrán en común los temas trabajados y que se defenderán en **coloquios**
- en el **desarrollo de un trabajo práctico final integrados**.

Metodología de evaluación: Promoción directa

TALLER DE LENGUAJES II

¿Qué vamos a aprender? ¿Cómo trabajaremos?
¿Qué herramientas usaremos?



TALLER DE LENGUAJES II

Tema de la clase de hoy:

Conceptos de Programación Orientada a Objetos

Programación Orientada a Objetos

Introducción

La **Programación Orientada a Objetos (POO)** es un paradigma de programación que **asocia comportamiento o acciones a estructuras de datos** llamadas **objetos** que pertenecen a **clases** y que están organizadas en **jerarquías**.

Principios de la POO:

- Abstracción
- Encapsulamiento y Ocultamiento de Información
- Herencia
- Polimorfismo

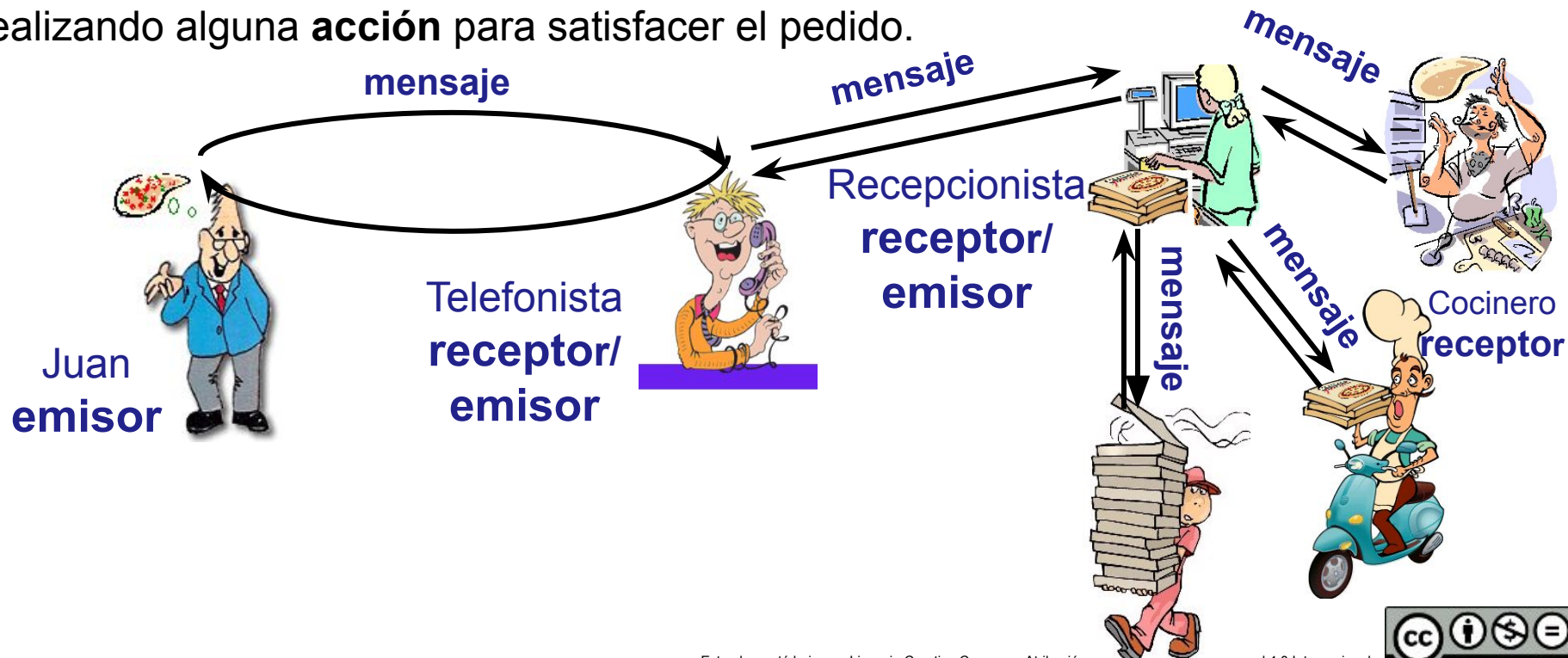
Existen múltiples lenguajes de programación orientados a objetos: **JAVA, Kotlin, Smalltalk, C++, Python, Ruby, C#, etc.**

Programación Orientada a Objetos

Objetos y Mensajes

Un programa en **POO** está organizado como un conjunto de **objetos** que colaboran para la **realización de una tarea**. Cada **objeto** se compone de **datos y operaciones o acciones** que manipulan esos datos. Cada objeto sabe responder **mensajes** que recibe de otros objetos.

Los **objetos se comunican** mediante **mensajes**. Un **mensaje** es un **pedido de realización de una acción** y está compuesto por un **objeto emisor** y un **receptor**. El **emisor** es el que inicia el mensaje y el **receptor** es el que resuelve el mensaje realizando alguna **acción** para satisfacer el pedido.

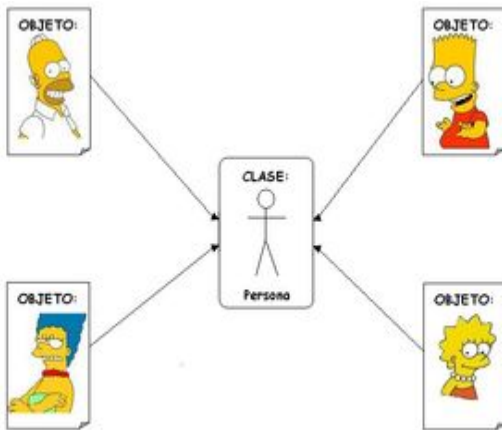


Programación Orientada a Objetos

Clases e Instancias

Una **clase** es una representación o modelo a partir del que se crean objetos o **instancias** de la clase.

Una clase **describe atributos** o **propiedades** y **comportamientos**, comunes a un grupo de objetos.



Clasificar significa que los **objetos con los mismos atributos y comportamiento** se agrupan en una **clase**.



Instanciar es la **acción de crear objetos a partir de una clase**, es por esto que a los objetos se los conoce como **instancias de clases**.

Cada **objeto** tiene su **propia identidad**: 2 objetos son distintos aún si los valores de sus propiedades son idénticas. **¿Qué significa?**

Los **objetos** contienen un **estado** o **datos propios** y un **conjunto de comportamientos** u **operaciones** que pueden realizar. Comparten nombres de atributos y de operaciones con el resto de los objetos de la clase.

Programación Orientada a Objetos

Abstracción

Las aplicaciones de software, en general, modelan situaciones o problemas del mundo real para automatizarlos. El **mundo real es complejo** a simple vista y cuando se lo observa con más detalle, la complejidad crece.

¿Cómo modelamos este mundo tan complejo?

Los humanos entendemos al mundo construyendo modelos mentales de partes del mismo. Un modelo mental es una **visión simplificada de cómo las cosas funcionan y cómo podemos interactuar con ellas**.

La **abstracción** es la capacidad de **aislar** o **ignorar** las **características**, propiedades o funciones **irrelevantes** y **enfaticar** en aquellas **cuestiones** que consideramos **relevantes**. *La abstracción nos permite gestionar la complejidad.*

“Una abstracción es una descripción simplificada o especificación de un sistema, que enfatiza algunos de los detalles o propiedades del mismo, mientras suprime otros. Una buena abstracción es aquella que enfatiza detalles significativos al lector o usuario y suprime detalles que son, al menos por el momento, irrelevantes o causan de distracción” (Mary Shaw, 1984).

Programación Orientada a Objetos

Abstracción

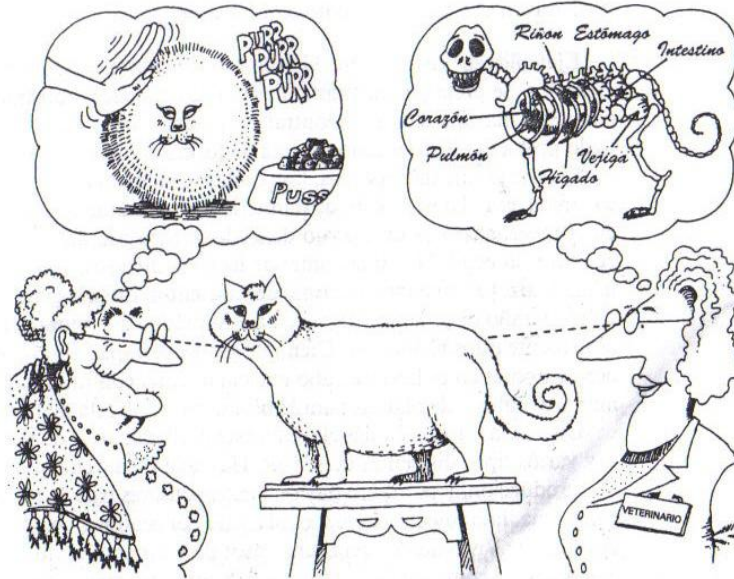


Imagen del libro "Object Oriented Analysis and Design" de Grady Booch

La abstracción se centra en las características esenciales de un objeto dependiendo del observador

En POO cada **objeto** del programa abstrae **datos** también llamados **estado del objeto** y **comportamiento** o las **acciones** que el objeto sabe hacer.

¿Qué características podemos abstraer de los autos?

Datos: marca, modelo, número de chasis, peso, llantas, puertas, ventanas, etc.

Comportamiento: acelerar, frenar, retroceder, etc.

¿Qué características podemos abstraer de los aves?

Datos: pico, alas, plumas, patas, etc

Comportamiento: volar, detenerse, etc.

Programación Orientada a Objetos

Encapsulamiento y Ocultamiento de Información

El **encapsulamiento** construye una cápsula o barrera alrededor de los datos y el comportamiento de los objetos. Establece: **“todas estas cosas deben permanecer juntas”**.

El **encapsulamiento** NO determina la visibilidad del interior de la cápsula.

El **ocultamiento de información** (*information hiding*) es la capacidad de ocultar o esconder los datos y comportamiento encapsulados. Establece qué forma parte de la **interfaz pública** y qué de la representación privada o **implementación**. Provee un efecto de “caja negra”.



Interfaz Pública

Implementación

El **encapsulamiento** y el **ocultamiento de información** se complementan para aislar las diferentes partes de un sistema, permitiendo que el código sea cambiado y extendido sin el riesgo de que deje de funcionar o de producir efectos colaterales no intencionados.

1. Se abstrae la funcionalidad y la información relacionada y se **encapsula** en clases.
2. Se decide qué operaciones e información podrá ser requerida por otros objetos y el resto se **oculta**.

Programación Orientada a Objetos

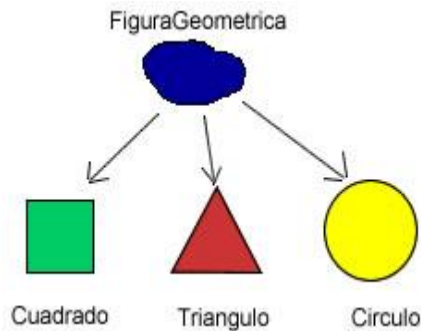
Herencia

La **herencia** es la propiedad que permite definir una nueva clase, **subclase** o **clase hija**, a partir de otra existente o **superclase**. Las subclases **heredan** las **propiedades** y el **comportamiento** de la **superclase**, pudiendo **reemplazar** y **añadir** nuevo **comportamiento**.

La **herencia** permite **compartir datos y comportamiento** entre **subclases similares** y así **evitar redundancias**.

La **reutilización de código** es el principal beneficio de la herencia.

La **claridad conceptual** de reconocer que operaciones diferentes son en realidad la misma cosa.



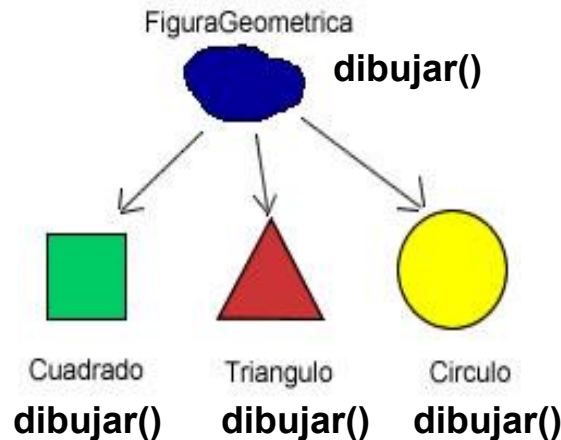
Ej: Un triángulo es una figura geométrica; tiene características comunes con otras figuras geométricas (cuadrado, círculo), por ejemplo el perímetro y la superficie.

Las clases pueden organizarse en **jerarquías de herencia** donde las **subclases** heredan **estado y comportamiento** de las clases que se encuentran más arriba en la jerarquía o **superclases**.

Programación Orientada a Objetos

Polimorfismo

El **polimorfismo** es la capacidad de tomar **diferentes formas**. El polimorfismo permite definir **diferentes comportamientos** para una **misma operación**, dependiendo del objeto receptor.



```
FiguraGeometrica[] f=new FiguraGeometrica[] {new Cuadrado(), new Triangulo(), new Circulo()};
```

En POO los **métodos** son implementaciones de las **operaciones**.

La clase **FiguraGeometrica** define el método **dibujar()** que contiene código común a todas sus subclases; las subclases **Cuadrado**, **Triangulo** y **Circulo** definen un **dibujar()** adaptado.

Cuando usamos el método **dibujar()** no tenemos que hacer ninguna distinción entre cuadrados, triángulos y círculos, el **polimorfismo** es el mecanismo que determinará **qué dibujar() se ejecutará** dependiendo del objeto receptor.

MODELAMOS UN PROBLEMA USANDO UN ENFOQUE OO

Supongamos que se quiere desarrollar un sitio de venta de bicicletas *on line*. ¿Cómo delimitamos el alcance del problema o dominio del problema?

- 1) Relevamiento de las necesidades de los usuarios -> Necesidades
- 2) ¿Cómo resolvemos el problema a partir de las necesidades de los usuarios con un análisis orientado a objetos?

- Descubrir o identificar las **entidades** del dominio del problema -> Objetos
- Identificar relaciones entre las entidades -> Taxonomía de Objetos

TIP: los nombres de los objetos generalmente son sustantivos, por ejemplo bicicleta, orden de compra, alumno, reloj, etc

3) ¿Cómo modelamos los objetos?

- Identificar las **características** relevantes de los objetos. Pensar en las partes que conforman el objeto, las cuales son generalmente reconocidas como “sustantivos”. Por ejemplo, las partes de una bicicleta podrían ser: cuadro, asiento, pedal, cadena, llantas, etc.
- Identificar las **operaciones** de los objetos. Son acciones que los objetos saben/pueden hacer. Generalmente son verbos o combinaciones de verbos y sustantivos, por ejemplo, las bicicletas saben arrancar, acelerar y frenar.

4) ¿Cómo podemos formalizarlo?

Lenguaje de modelado UML.

IDENTIFICAR OBJETOS

Podemos mirar a nuestro alrededor y observar muchos objetos del mundo real: un perro, un escritorio, un televisor, una bicicleta, una computadora, etc. Estos son objetos físicos que podemos “tocar”; también hay objetos abstractos como pueden ser una orden de compras, una cuenta bancaria, el salario, etc.

Para **determinar los objetos del dominio de nuestro problema**, es necesario analizar su **relevancia** dentro del problema.

Una vez identificados los objetos, debemos identificar sus **atributos** y **operaciones relevantes para el problema**.

Los **atributos** son las características relevantes de los objetos en el problema a resolver. Por ejemplo podrían ser tamaño, forma, color, etc. A los **valores** de todos los **atributos** de un objeto se los denomina **estado del objeto**. Por ejemplo, un objeto podría tener el valor “rojo” en el atributo **color** e “irregular” en el atributo **forma**.

Las **operaciones** son las acciones que los objetos saben hacer. Por ejemplo una figura geométrica sabe dibujarse en la pantalla o un auto sabe acelerar a una velocidad, una orden de compras sabe imprimirse, agregar/quitar un producto. **Las operaciones pueden modificar los valores de los atributos**. A todas las operaciones que un objeto puede realizar se las denomina **comportamiento del objeto**. Por ejemplo una figura geométrica podría tener una operación que cambie su color, otra que calcule una distancia a otra figura, otra que la dibuje en la pantalla, etc.

IDENTIFICAR ATRIBUTOS Y OPERACIONES

Objetos concretos

Características de una bicicleta:

atributos: cuadro, asiento, pedal, color, llantas, etc.

comportamiento: acelerar, realizar un cambio, frenar, etc.

estado de una bicicleta particular: cuadro de aluminio, pedal doble, color rojo, llantas de aluminio, asiento ergonómico, etc



Características de un perro:

atributos: nombre, color, raza, etc.

comportamiento: ladrar, correr, jugar, etc.

estado de un perro particular: nombre RITA, color negra, raza galgo, etc



Objetos abstractos

Características de una tarjeta SUBE:

atributos: numero, titular, saldo, etc.

comportamiento: cargar, consultarSaldo, pagarBoleto, etc.

estado de una SUBE particular: número 6061 3456 7890 0056, titular Juan Pérez, saldo \$200

RESUMIENDO....

- Desde el enfoque de OO los objetos permiten modelar problemas del mundo real.
- Resolver un problema usando un análisis orientado a objetos requiere identificar los objetos de acuerdo a su relevancia en el dominio del problema y las relaciones entre ellos.
- Los objetos se caracterizan por tener **estado** y **comportamiento**.

EJEMPLO DE MODELADO OO

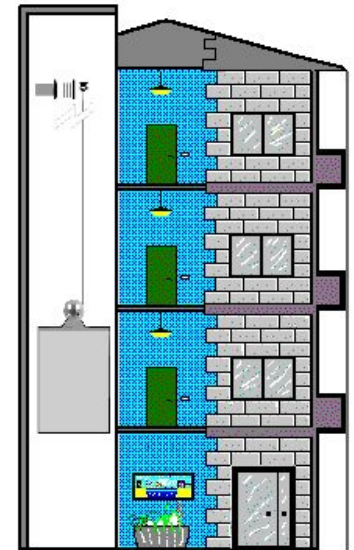
Se quiere modelar el uso de un ascensor en un edificio.

El edificio tiene 3 pisos en los que habitan 9 familias. El ascensor es usado por todas aquellas personas que ingresan al edificio.

¿Qué objetos podemos identificar?



- Personas
- Ascensor
- Departamento
- Edificio
- Portón de entrada
- Ventanas
- y muchos más...



RECONOCER LOS OBJETOS DEL DOMINIO DEL PROBLEMA

Es importante **identificar a los objetos relevantes** y quedarnos solamente con aquellos que son de nuestro interés para el problema que estamos modelando.

Dependiendo del contexto y punto de vista de quien modela, es el resultado del modelo obtenido.

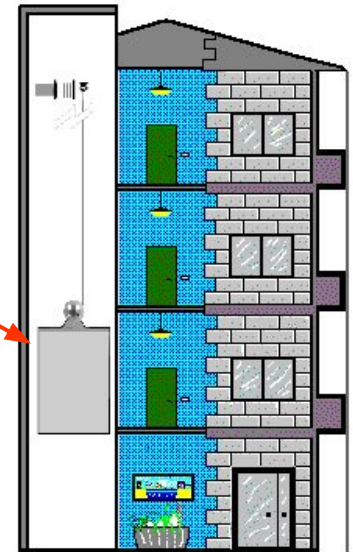
Hay que tener en cuenta la relevancia para el dominio del problema.

IDENTIFICAMOS LOS OBJETOS DE NUESTRO PROBLEMA

Para nuestro problema sólo son relevantes los siguientes objetos:



- Persona
- Ascensor
- ~~Departamento~~
- ~~Edificio~~
- ~~Portón de entrada~~
- ~~Portero~~
- ~~Ventanas~~



IDENTIFICAMOS LOS ATRIBUTOS Y EL COMPORTAMIENTO

¿Cómo describimos al objeto Persona?



Atributos

nombre
apellido
edad
domicilio
documento

Comportamiento

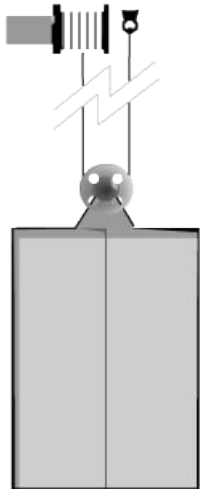
sabe decir cómo se llama
sabe decir cuántos años tiene
sabe en qué piso vive
sabe su número de Documento

Con esta información se va a poder comunicar
con el ascensor y decirle a donde quiere ir

Buena práctica: utilizar nombres de atributos y de operaciones que describan claramente el atributo y la operación.

IDENTIFICAMOS LOS ATRIBUTOS Y EL COMPORTAMIENTO

¿Cómo describimos al objeto Ascensor?



Atributos

- piso actual
- estado puerta (abierta / cerrada)
- cantidad de ocupantes
- máxima cantidad de ocupantes
- piso Máximo
- piso Mínimo

Comportamiento

- subir
- bajar
- abrir la puerta
- cerrar la puerta
- llamar

INTERACCIÓN ENTRE OBJETOS

Los objetos tienen **estado** y **comportamiento** pero
¿Cómo se comunican entre ellos?

Enviándose mensajes

Consideremos a una persona, María, que usa el ascensor de su edificio
para llegar a su departamento.

¿Cuáles son los mensajes que María utiliza para comunicarse
con el ascensor de su edificio?



1 – Llamar

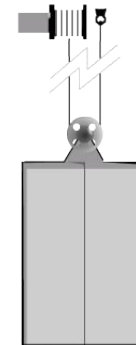
2 – Abrir la puerta

3 – Cerrar la puerta

4 – Subir/Bajar (piso)

5 – Abrir puerta

6 – Cerrar puerta

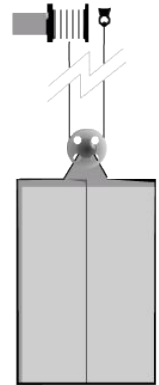


INTERACCIÓN ENTRE OBJETOS

En esta secuencia podemos observar la interacción entre los dos objetos del modelo que hemos identificado:

María se comunica con el ascensor de su edificio:

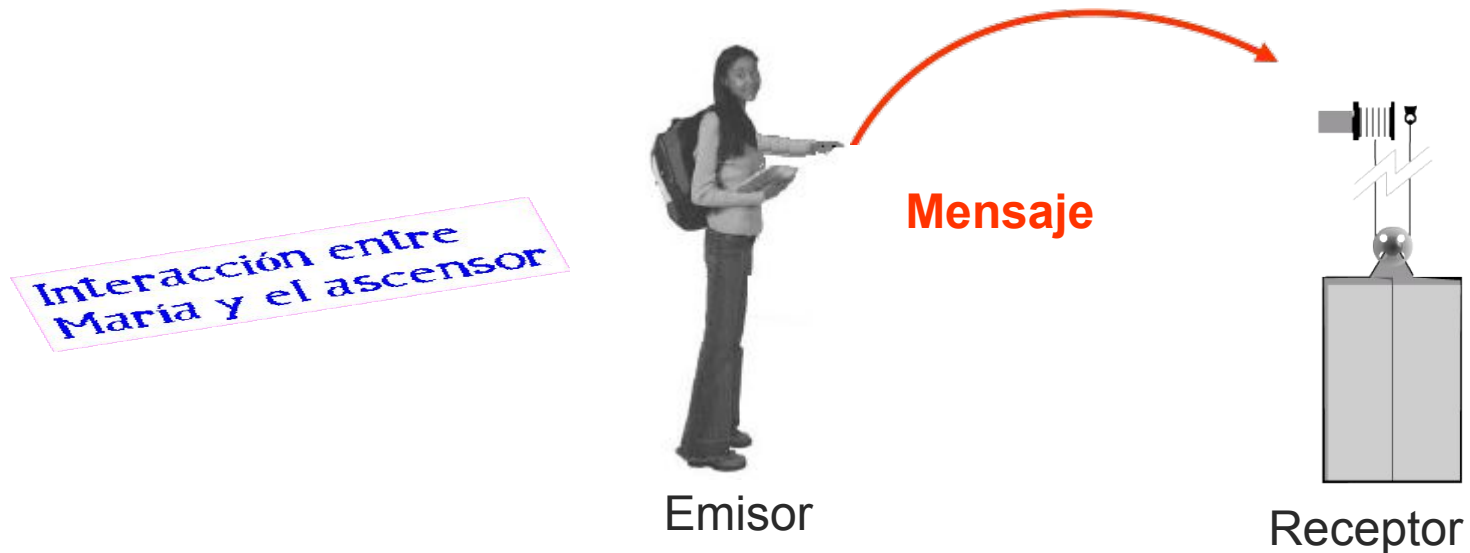
- 1 A través de un **mensaje**, lo **llama**
- 2 El **ascensor** tiene la **responsabilidad** de responder el mensaje yendo al piso desde el que **María** lo llamó.
- 3 **María** sube al ascensor y a través de un nuevo **mensaje** le dice que la lleve a un determinado piso.
- 4 El **ascensor** tiene la **responsabilidad** de llevar a **María** al piso indicado.



INTERACCIÓN ENTRE OBJETOS

¿Qué hace María para llegar hasta su departamento?

- 1 Pararse frente a la puerta del ascensor y **llamarlo**.



INTERACCIÓN ENTRE OBJETOS

- 2 El **ascensor** llega al piso, **abre** la puerta, entra **María**. El ascensor espera que María le indique a **qué piso quiere ir**.



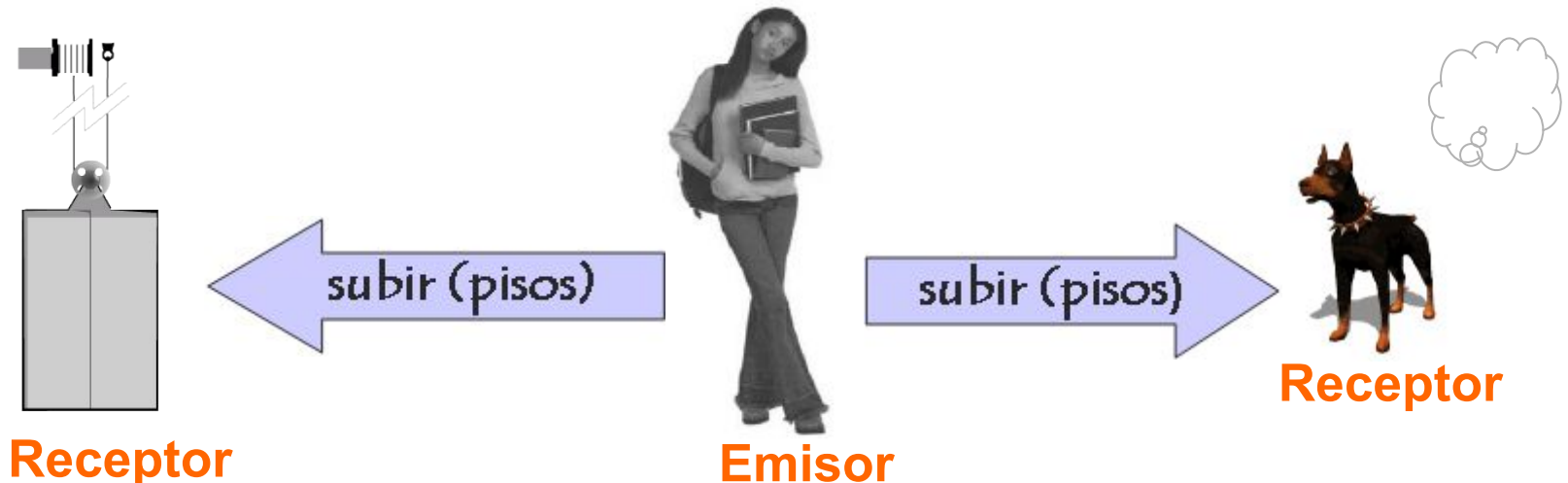
- 3 **María** le indica a qué piso quiere ir, entonces el ascensor **cierra** la puerta y **sube o baja** hasta el piso seleccionado.

MENSAJES

Métodos vs. Procedimientos

Existen 2 distinciones importantes:

- 1 En un **mensaje** siempre hay designado un **receptor** del mensaje: es el objeto al que se le envía el mensaje. Cuando se invoca a un **procedimiento** **NO** hay un **objeto receptor**.
- 2 La **ejecución del mensaje** es llevada a cabo por un **método** quien responde el mensaje. El **objeto receptor** determina **qué método será el encargado de responder el mensaje** y podría variar para diferentes receptores.



CLASES E INSTANCIAS

María al igual que otras personas que toman ascensores tienen características comunes: pueden **llamar a un ascensor**, indicarle **a qué piso desean ir**, etc. esto es porque pertenecen a una Categoría o Clase que podríamos llamar "Usuario".

Una clase es un modelo a partir de la cual se crean instancias con las mismas características y comportamiento.



María, Juan, Pedro, Catalina y otras personas que hacen uso del ascensor son **instancias** de la **clase** Usuario.

Aquí tenemos una categoría general, **Personas** y otra específica **Usuario**

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

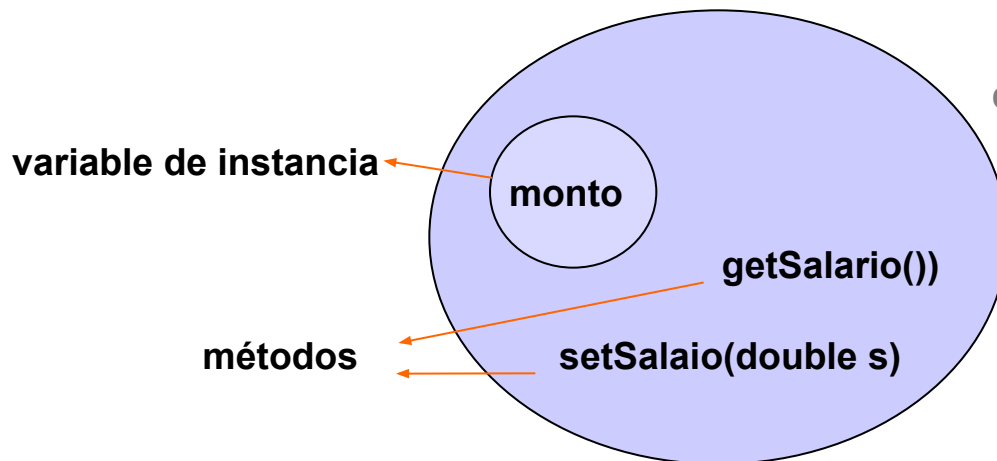
¿Cómo implementa JAVA el ENCAPSULAMIENTO?

Los lenguajes de programación orientada a objetos proveen construcciones llamadas **clases** que permiten agrupar objetos con las mismas características.

Una clase JAVA permite agrupar los atributos o variables de instancia y el comportamiento o métodos de los objetos en clases. **Las clases JAVA proveen encapsulamiento.**

Consideremos la clase **Salario**

ENCAPSULAMIENTO: VARIABLES DE INSTANCIA + MÉTODOS



Cambia la fórmula de liquidar sueldos, se debería modificar el método `getSalario()`, pero los programas que lo usan no se deberían ver afectados por el cambio.

El **encapsulamiento** provisto por las clases NO determina la visibilidad del interior de los objetos.

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

¿Cómo implementa JAVA el OCULTAMIENTO DE LA INFORMACIÓN?

El **ocultamiento de la información** es la capacidad de ocultar los detalles internos de un objeto y exponer sólo los que son necesarios para el resto del sistema. De esta manera las modificaciones que realicemos no afectarán a quienes usan nuestras clases.

Se denomina **interface pública** a las partes de los objetos que conoce el mundo exterior; está compuesta por los métodos y los datos que podrán ser requeridos directamente por otros objetos del sistema. Los restantes datos y métodos forman parte de la representación privada de los objetos o **implementación**.

En JAVA el ocultamiento se implementa aplicando los modificadores de acceso en las declaraciones de las variables y métodos definidos en las clases.

En el ejemplo de la **clase Salario** los métodos **getSalario()** y **setSalario()** pertenecen a la **interface pública** de los objetos Salario y el **monto** es parte de la **implementación**.

La palabra clave **private** que se antepone a la declaración de las variables de instancia y de los métodos permite implementar **ocultamiento** (el más estricto). Sin embargo, JAVA provee una gama más amplia de modificadores de acceso que desarrollaremos en el curso y que permite definir diferentes formas de ocultamiento.

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

¿Cómo implementa JAVA la HERENCIA?

Además de contar con cierta información sobre María por ser un **Usuario**, sabemos que María también es una **Persona**, que es un **Mamífero** y también, es un **Ser Vivo**. Organizamos el conocimiento como una jerarquía de clases. En nuestra solución NO necesitamos una jerarquía tan amplia.

Todo este conocimiento que tenemos sobre María también puede aplicarse a otros usuarios.

En JAVA la herencia es simple: una clase sólo puede heredar de una superclase.

JAVA provee la palabra clave **extends** para declarar que **una clase es subclase de otra clase**.

JAVA implementa la herencia a través del **encadenamiento de constructores**: **el objeto de la subclase contiene un subobjeto de la clase base que se crea en el momento de la instanciación.**

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

¿Cómo implementa JAVA el POLIMORFISMO?

El polimorfismo permite definir **diferentes comportamientos para un mismo método**, dependiendo del objeto receptor.

El polimorfismo en JAVA se resuelve mediante el ***binding dinámico ó late binding (vínculo tardío)***, esto significa que la conexión entre la invocación a un método y el cuerpo o código del método se resuelve en ejecución.

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Síntesis

1. **Todo es un objeto.** Un programa orientado a objetos está organizado como un conjunto de objetos que interactúan mediante el envío de mensajes.
2. La computación se lleva a cabo por **objetos que interactúan entre sí enviándose mensajes.**
3. **Todos los objetos son instancias de una clase.**
4. **Una clase es un modelo con una estructura y comportamiento asociado** que tendrán todos los objetos que con ella se creen. Todos los objetos que se crean con una clase tendrán los mismos atributos con diferentes valores (estado) y podrán ejecutar los mismos métodos (comportamiento).
5. El **encapsulamiento y el ocultamiento de información** se complementan para aislar las diferentes partes de un sistema, permitiendo que el código sea modificado y extendido sin producir efectos colaterales. En Java está implementada a través de clases (encapsulamiento) y modificadores de acceso (ocultamiento).
6. Las clases son organizadas en taxonomías que permiten relacionar clases y proveen **herencia** de atributos y comportamiento. En JAVA la herencia es simple, es una estructura jerárquica llamada **jerarquía de herencia**. Los atributos y el comportamiento asociados con una clase están automáticamente disponibles para sus subclases.
7. El método invocado por un objeto en respuesta al envío de un mensaje es determinado por el objeto receptor del mensaje. **Polimorfismo**

MODELAMOS LAS CLASES DE NUESTRO PROBLEMA

Persona
<ul style="list-style-type: none">- nombre: String- apellido: String- edad: Integer- documento: String
<ul style="list-style-type: none">+ setEdad()+ getEdad()+ getNombre()+ getApellido()

↑ Es_Un

Usuario
<ul style="list-style-type: none">- piso: Integer
<ul style="list-style-type: none">+ setPiso(x)+ getPiso()+ irAPiso(x)

Detalles privados
de Implementación

Interface
pública

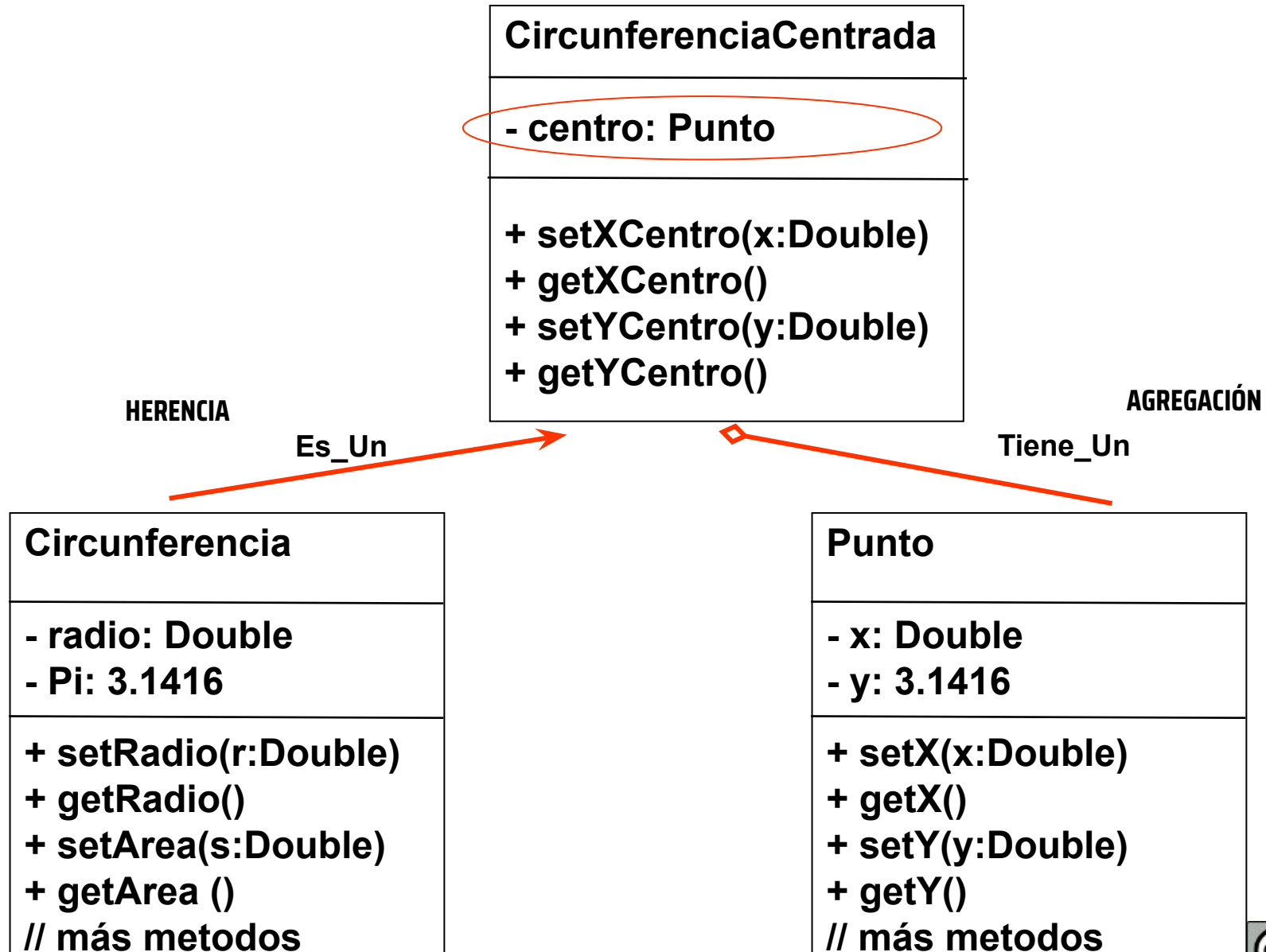
maria.irAPiso(2);

irAPiso(x)



Ascensor
<ul style="list-style-type: none">- pisoActual: Integer- estadoPuerta: Boolean- capacidad: Integer- pisoMaximo: Integer- pisoMinimo: Integer
<ul style="list-style-type: none">+ subir()+ bajar()+ abrirPuerta()+ getPisoActual()+ setPisoActual()+ getCapacidad()//más métodos

miAscensor.abrirPuerta()
miAscensor.getPisoActual()
.....
miAscensor.subir(x)

RELACIONES ENTRE CLASES



RELACIONES ENTRE CLASES

HERENCIA	Relación de “especialización/generalización” entre objetos. Relación “es-un” .	
AGREGACIÓN	Un objeto como composición de otros objetos. Relación “todo-partes” La clase que estamos definiendo “Tiene-un” atributo de otras clase. Los objetos que representan las “partes” pueden existir independientemente de la clase “todo”.	
COMPOSICIÓN / AGREGACIÓN FUERTE	Los objetos agregados (“partes”) no tienen pueden existir independientemente de la clase “todo”.	