Unidad 1

# El Modelo Orientado a Objetos


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# El Modelo Orientado a Objetos

## Unidad 1

- **OBJETIVOS**
  - Reconocer los conceptos y las diferencias entre objetos y clases.
  - Comprender los principios fundamentales de la OO.
  - Entender las diferencias entre el paradigma estructurado y el OO.

# El Modelo Orientado a Objetos

## Unidad 1

- **HABILIDADES Y COMPETENCIAS QUE DESARROLLA LA ASIGNATURA**
- Lograr analizar y diferenciar paradigmas estructurados frente a los paradigmas orientados a objetos.
- Adquirir conocimientos de análisis para los principios fundamentales de la OO.



# Introducción:
# Consideraciones Generales


UAI
Universidad Abierta Interamericana

UAIOnline Ultra

# EL PARADIGMA OO SE IMPUSO POR…

- Conceptos comunes de modelado a lo largo de todo el ciclo de vida
- Reducción de la brecha entre el mundo de los problemas y el mundo de los modelos
- Aumento de complejidad de los sistemas
- Aumento de la necesidad de reutilización
- Uso de patrones

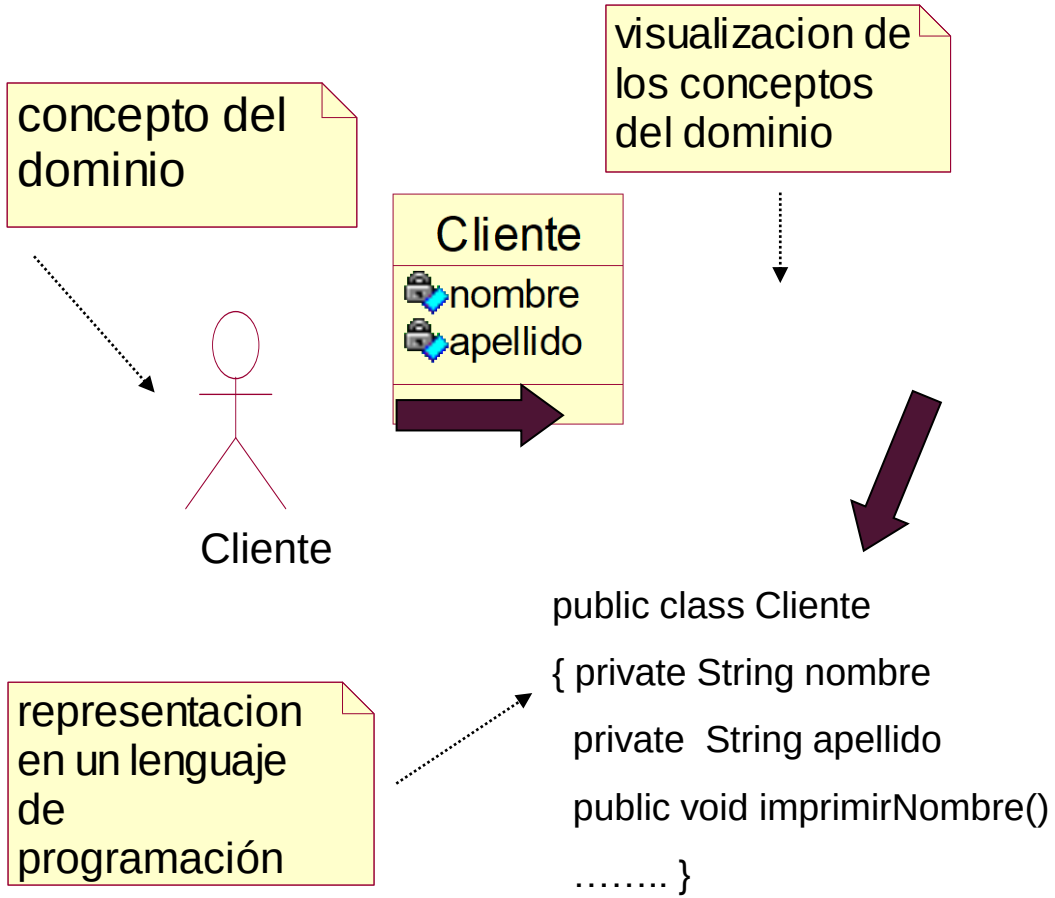
# ANÁLISIS, DISEÑO, IMPLANTACIÓN

- El **análisis** OO pone énfasis en la investigación del problema y los requisitos, en vez de ponerlo en la solución
- El **diseño** pone énfasis en una solución conceptual, que satisface los requisitos, en vez de ponerlo en la implantación
- La **implantación** es la traducción de la solución a un lenguaje de programación determinado

# ANÁLISIS OO VS. DISEÑO OO

- Durante el análisis OO se presta especial atención a encontrar y describir los objetos (conceptos) del dominio del problema

- Durante el diseño OO se presta atención a la definición de los objetos software y en como colaboran para satisfacer los requisitos



# ANÁLISIS OO

- La finalidad del análisis OO es crear una descripción del dominio desde una perspectiva de clasificación de objetos: identificación de conceptos, atributos e interrelaciones significativas

- El modelo del dominio **NO** es una descripción de los objetos software, es una visualización de los conceptos del mundo real y sus vinculaciones (se representan mediante diagrama de clases, sin operaciones)

# ABSTRACCIÓN Y ENCAPSULAMIENTO

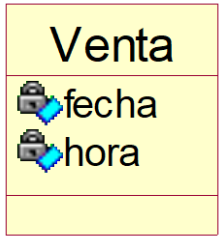
- La **abstracción** es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales).

- El **Encapsulamiento** es la propiedad que permite asegurar que el contenido de la información de un objeto está oculta al mundo exterior .

- El encapsulamiento, al separar el comportamiento del objeto de su implantación, permite la modificación de éste sin que se tengan que modificar las aplicaciones que lo utilizan

# CLASES CONCEPTUALES

- Una clase conceptual se puede considerar en términos de:




- ***Símbolo***: palabras o imágenes que representan la clase conceptual

Una venta representa una transacción de compra

- ***Intensión***: la definición de la clase conceptual

- ***Extensión***: el conjunto de ejemplos a los que se aplica la clase conceptual

venta 1
venta 3
venta 2

# MODELO DEL DOMINIO

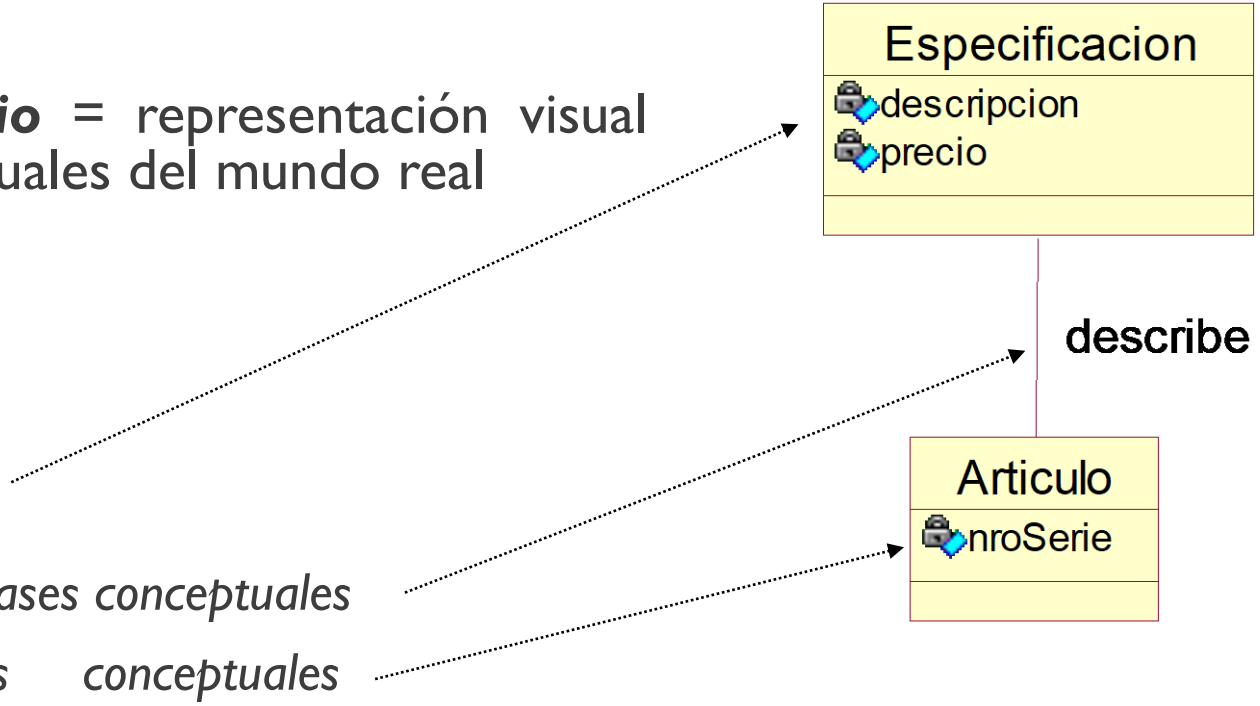
- ***Análisis*** = descomposición de un dominio de interés en clases conceptuales

- ***Modelo del dominio*** = representación visual de las clases conceptuales del mundo real

Se visualizan en

el modelo de dominio:

- *Clases conceptuales*
- *Asociaciones entre clases conceptuales*
- *Atributos de las clases conceptuales*



# RESPONSABILIDADES

- Una responsabilidad es un contrato u obligación de una clase
- Una clase puede tener cualquier numero de responsabilidades, pero una clase bien estructurada debería tener al menos una y como mucho unas pocas (alta cohesión)
- Las responsabilidades se describen inicialmente con texto libre
- Al ir refinando los modelos, las responsabilidades se traducirán en el conjunto de atributos y operaciones que satisfagan esas responsabilidades

# TARJETAS CRC (COLABORACIÓN-RESPONSABILIDAD-CLASE)

Por cada clase describimos:

El nombre de la clase y su descripción

La responsabilidad de la clase

- Conocimiento interno de la clase
- Servicios brindados por la clase

Los colaboradores para las responsabilidades

- Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad

Usaremos una adaptación de las tarjetas CRC como primera aproximación para luego refinar las clases en la etapa de diseño

# TARJETAS CRC (COLABORACIÓN-RESPONSABILIDAD-CLASE)

- El nombre de la clase: es importante que el nombre refleje la esencia de lo que hace la clase.
- Las responsabilidades de la clase: determinan qué debe hacer. Estas responsabilidades pertenecen, esencialmente, a dos categorías: hacer y conocer.
- Las colaboraciones de la clase especifican con qué otras clases interactúan.

***HACER***

Hacer algo uno mismo.

Iniciar una acción en otros objetos.

Controlar y coordinar actividades en otros objetos

***CONOCER***

Conocer los datos privados encapsulados.

Conocer los objetos relacionados.

Conocer las cosas que se pueden derivar o calcular.

# TARJETAS CRC (ADAPTACIÓN)

**Ejemplo**

Nombre de la clase: **CURSO**
Descripción: específica el curso….

| | RESPONSABILIDADES | COLABORACIONES |
|---|---|---|
| HACER | Agregar un estudiante | **ESTUDIANTE** |
| | Conocer los prerrequisitos | |
| CONOCER | Conocer cuándo el curso es dado | |
| | Conocer dónde es dado el curso | |
| | | |

# DISEÑO OO

- La finalidad del diseño OO es definir los objetos software y sus colaboraciones (se utilizan en esta etapa diagramas de interacción: comunicación y secuencia)

- A diferencia del modelo del dominio, este modelo no muestra conceptos del mundo real, sino clases software, con atributos operaciones y asociaciones

# OBJETOS

"*Un objeto es cualquier cosa real o abstracta, acerca de la cual almacenamos datos y las operaciones que controlan dichos datos*"

Se opone al análisis estructurado donde los datos y el comportamiento están débilmente relacionados

Tenemos que olvidarnos del modelo estructurado...

# PROPIEDADES DE LOS OBJETOS

"El ***estado*** de un objeto abarca todas las propiedades (normalmente estáticas) del mismo, más los valores actuales (normalmente dinámicos) de cada una de esas propiedades"

"El ***comportamiento*** nos muestra como actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes"

"La ***identidad*** es aquella propiedad de un objeto que lo distingue de todos los demás objetos"

# CLASES

Una ***clase*** especifica una estructura de datos y las operaciones permisibles que se aplican a cada uno de sus objetos.

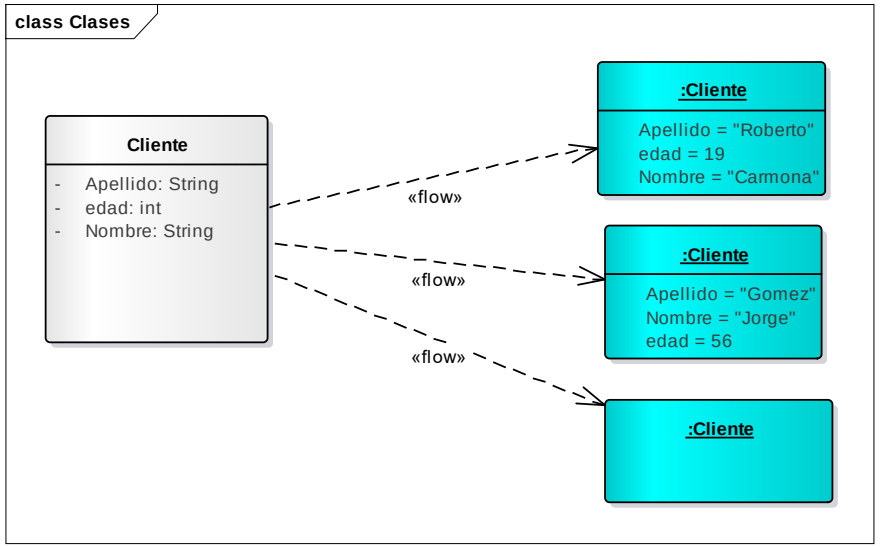
Los objetos se vinculan mediante enlaces enviando **mensajes** a **operaciones** que activan los **métodos**

- ***Mensaje***: es una solicitud para que se lleve a cabo la operación indicada y se produzca el resultado.
- ***Operaciones***: es una función o transformación que se aplica a un objeto de una clase
- ***Métodos***: es la implementación de una operación

"Un objeto es una instancia de una ***clase***"

# CLASES Y OBJETOS

- Posee una interfaz para acceder a sus miembros internos
- Una Clase Define (por medio del encapsulamiento) la forma de modificar el estado de sus objetos
- A los objetos se los conoce por su interfaz (tipo de datos)




¿Cómo modificamos el estado de estos objetos?

# RELACIONES DE ASOCIACIÓN

- "Se descompone (***clases***) para comprender, se une (***asociación***) para construir"
- Los enlaces entre objetos son instancias de la ***asociación*** entre sus clases
- La asociación representa un acoplamiento débil, la ***Agregación*** y la ***Composición*** expresa un acoplamiento más fuerte en clases

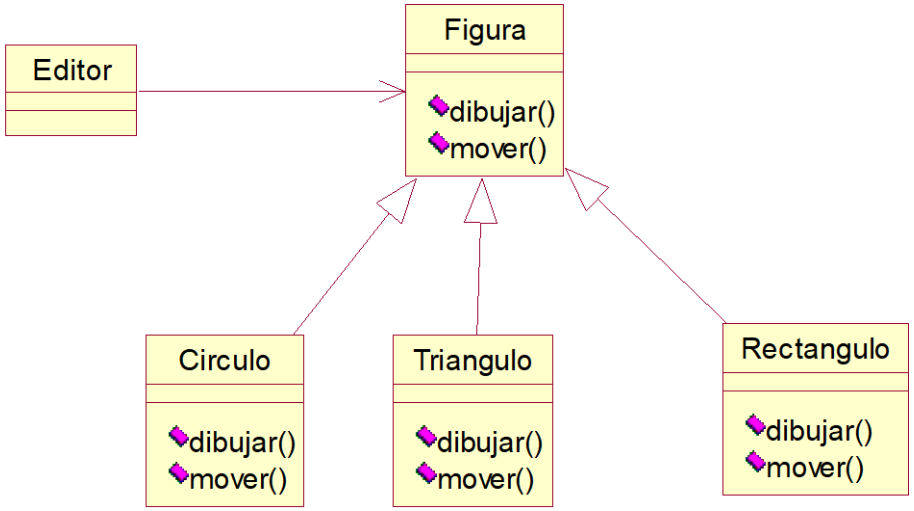
# RELACIONES DE JERARQUÍA

- La ***generalización*** consiste en factorizar los elementos comunes de un conjunto de clases en una clase más general llamada superclase

- La ***herencia*** es una técnica de los lenguajes de programación para construir una clase a partir de una o varias clases, compartiendo atributos y operaciones

# POLIMORFISMO/1

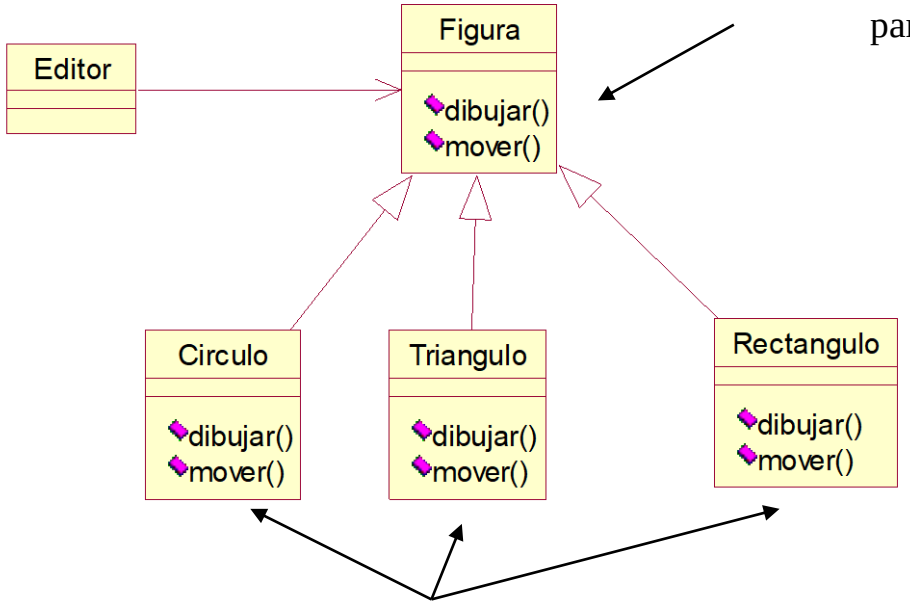
Permite la posibilidad de desencadenar operaciones diferentes en respuesta a un mismo mensaje




El polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento.

# POLIMORFISMO/2

La interacciones entre objetos se escriben según los términos de las especificaciones definidas en las superclases




El método dibujar() no es igual para cada figura

Cada vez que se invoque el método dibujar() dependerá de la clase a la pertenece el objeto

unCirculo.dibujar();

UnTriangulo.dibujar();

unRectangulo.dibujar();

Redefino en las subclases el método dibujar()

# ANÁLISIS ESTRUCTURADO VS. ANÁLISIS ORIENTADO A OBJETOS

- El enfoque tradicional del análisis y diseño estructurados, se descompone el problema en funciones o procesos y estructuras de datos

- En un enfoque OO se busca descomponer el problema, no en funciones, sino en unidades más pequeñas denominadas objetos

# BENEFICIOS DEL ENFOQUE OO

Disminución del bache semántico entre análisis y diseño proveyendo una representación consistente en todo el ciclo de vida

Enfoque OO

- La transición del análisis al diseño es un refinamiento

Enfoque Estructurado

- En la transición del análisis al diseño
- pasamos del DFD al DE mediante un proceso heurístico no trivial

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

- Single Responsability  Principle
- Open/Close Principle
- Liskov Sustitution Principle
- Interface Segregation Principle
- Dependency Inversión Principle

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

- Se los considera los cinco principios básicos en el diseño y la programación orientada a objetos

- La intención es aplicar estos principios en conjunto para que sea más probable obtener un software fácil de mantener y extender en el tiempo

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

*Single Responsability*

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Single Responsability:***

- Una clase debe tener solo una razón para cambiar
- Se centran en la Cohesión


# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

*Open/Close*

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Open/Close***

- El comportamiento de una entidad debe poder ser alterado sin tener que modificar su propio código fuente
- Una clase no se puede modificar, pero si se puede extender haciendo uso de la herencia
- Una clase solo debe ser modificada si existe un bug, no cada vez que se necesita agregar una nueva funcionalidad porque se puede romper alguna funcionalidad existente en módulos dependientes
- Ayuda a mantener bajo el acoplamiento
- Requiere mucho esfuerzo al diseñar las clases

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Open/Close***

- Una clase debe estar abierta para extenderla pero cerrada para modificarla
- Según el ejemplo, para administrar los empleados habría que modificar la clase Manager por cada nuevo empleado

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Liskov Substitution***

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Liskov Substitution:***

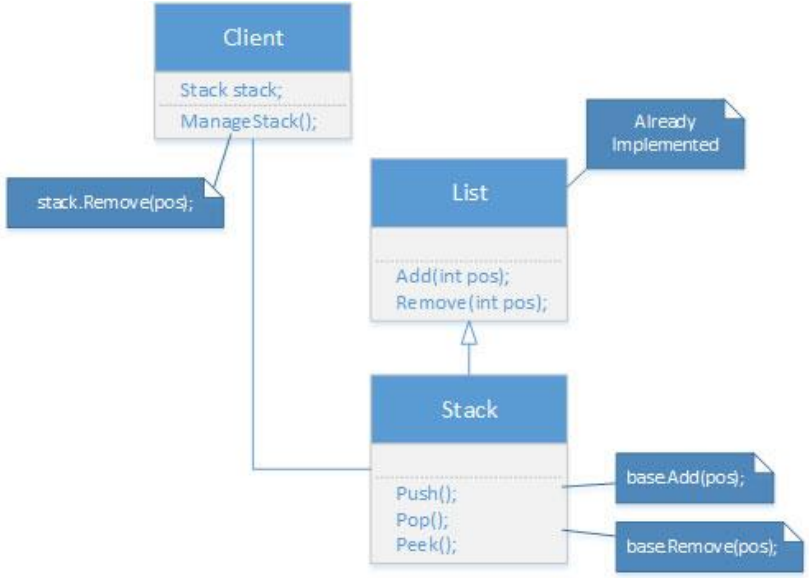
- Este principio es una extensión del principio Open Closed y señala que una clase derivada puede ser reemplazada por cualquier otra que use su clase base sin alterar el correcto funcionamiento de un programa.

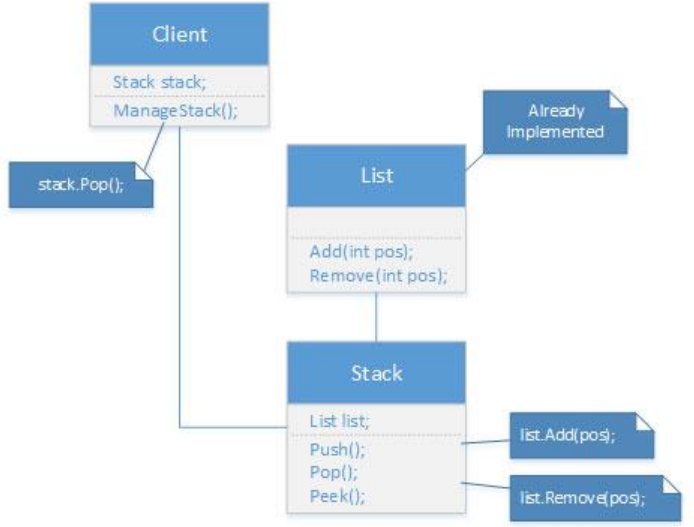
- si una función espera como parámetro una clase base esta puede ser reemplazada por cualquier clase derivada. Para no alterar el adecuado funcionamiento de un programa, una subclase no debe remover comportamiento de la clase base, no debe conocer a los demás subtipos.

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Liskov Substitution***

- Los tipos derivados deben estar completamente sustituibles por su tipo base
- En el ejemplo: que pasa si se ejecuta Remove()? Corresponde a List y no a Stack. Deberia tener una referencia a la clase BASE y no conocer el subtipo.

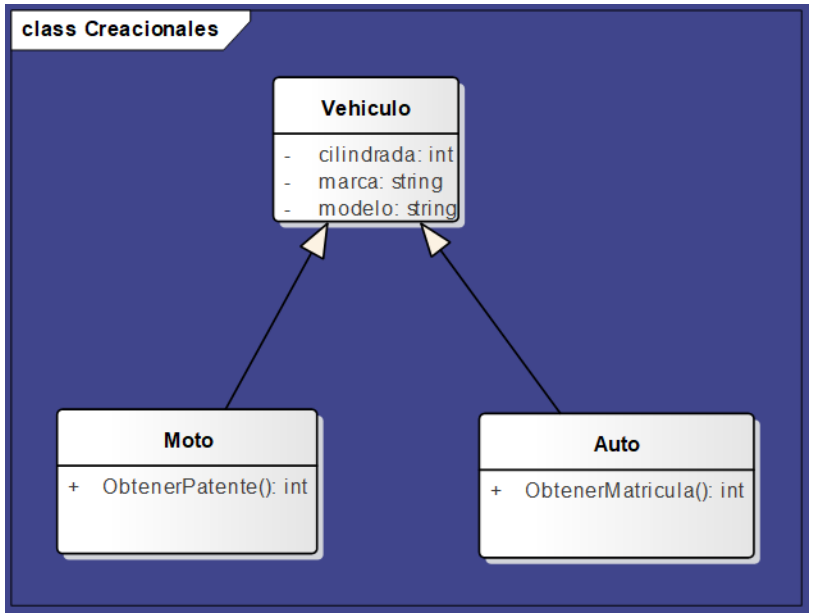


# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

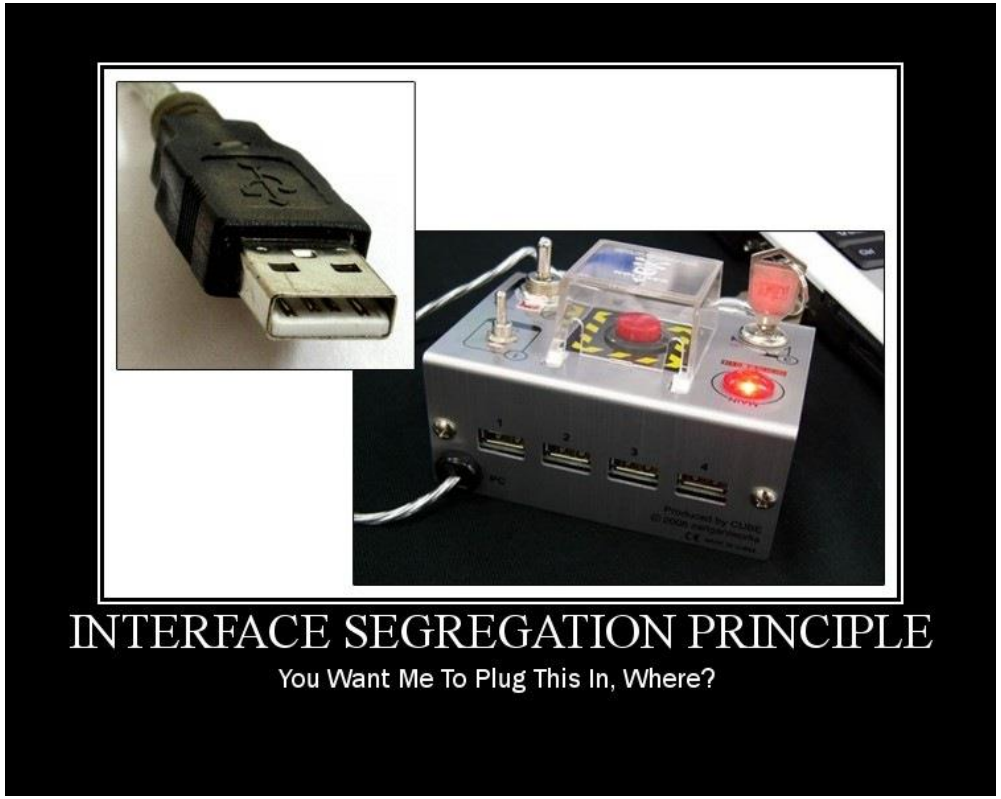
***Liskov Substitution  Por que se viola LSP?***

LSP afirma que si tenemos dos objetos de tipos diferentes –Coche y Ciclomotor– que derivan de una misma clase base –Vehículo–, deberíamos poder reemplazar cada uno de los tipos –Coche/Ciclomotor y viceversa– allí dónde el tipo base –Vehículo– esté implementado.



# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Interface Segregation***



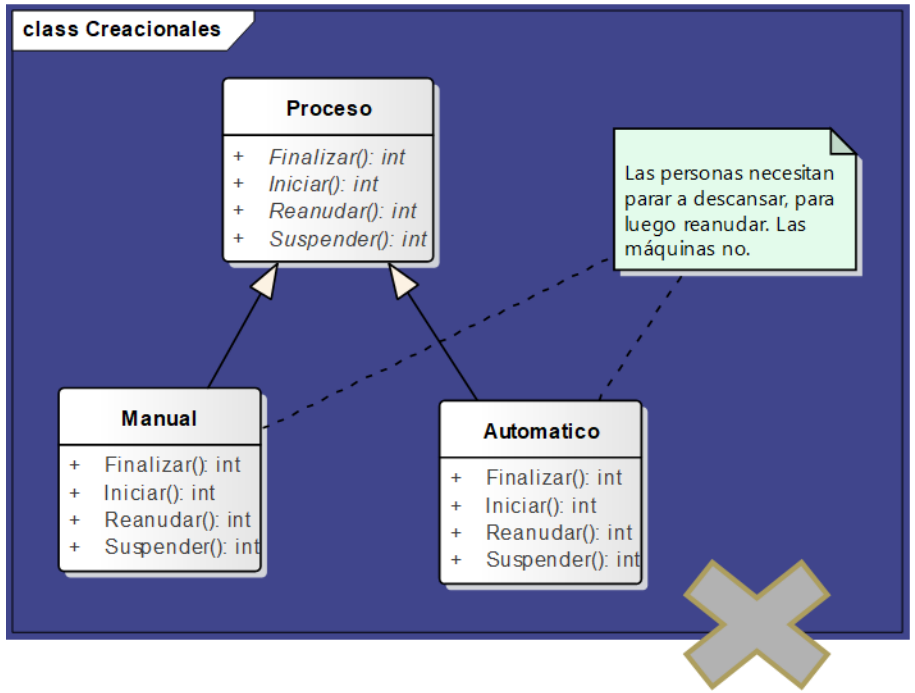
# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

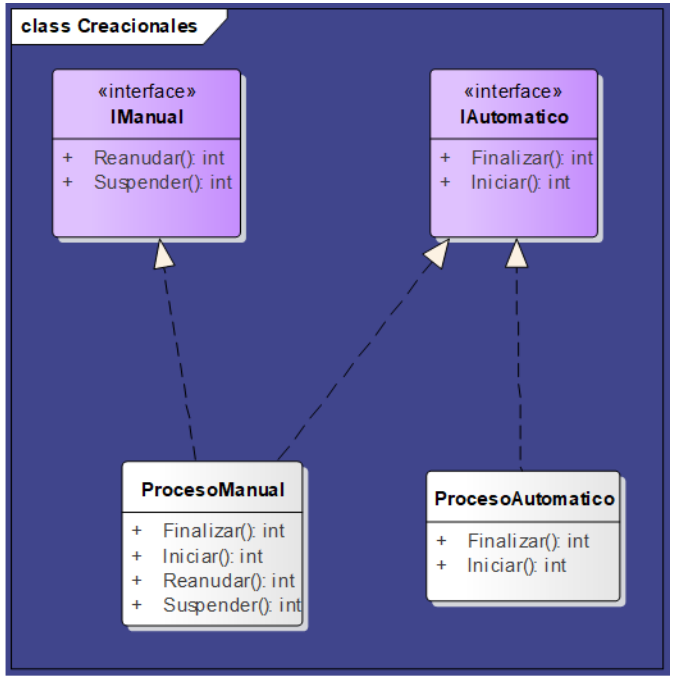
***Interface Segregation***

- Los clientes no deben estar forzados a implementar interfaces que no usan.
- Guarda relación con la cohesión de las aplicaciones.
- las clases que implementen una interfaz o una clase abstracta no deberían estar obligadas a utilizar partes que no van a utilizar
- Los clientes no deben estar obligados a implementar y/o a depender de una interface que luego no usarán.

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

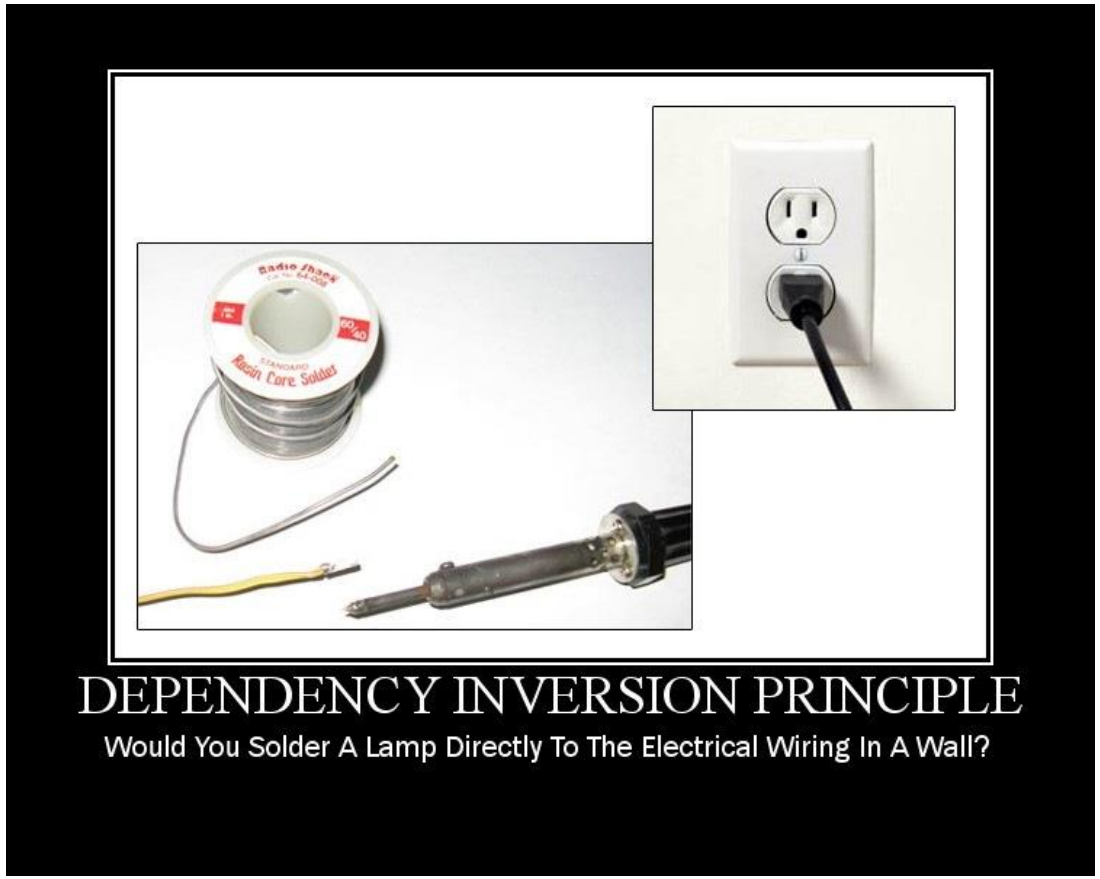
## *Interface Segregation*



# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

*Dependency Inversion*



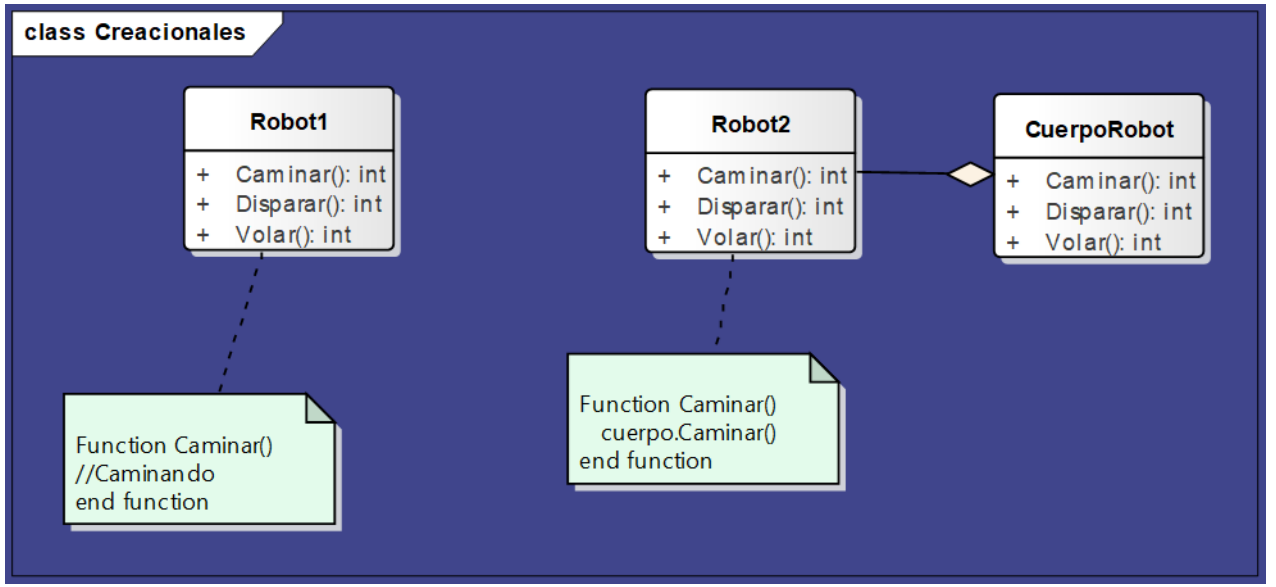
# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

***Dependency Inversion***

- Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
- Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.
- En el ejemplo Manager es de alto nivel y Worker es de bajo nivel. No hay dependencia

# 5 PRINCIPIOS BÁSICOS DEL REUSO SOLID

## *Dependency Inversion*



# SUGERENCIAS

- Cuando realice el ***análisis*** ponga énfasis en la investigación del problema y los requisitos
- Cuando realice el ***diseño*** ponga énfasis en una solución conceptual del problema
- En la etapa de análisis se realiza el ***Modelo del dominio*** (representación visual de las clases conceptuales del mundo real)
- Realice una tarjeta CRC para comenzar a entender la clase propuesta (esta será refinada posteriormente en la etapa de diseño)
- Si bien son conceptos vinculados, tenga presente la diferencia entre ***mensaje, operación y método***
- Recuerde que la ***composición*** y la ***agregación*** son tipos particulares de asociaciones
- Tenga en cuenta que la ***generalización*** es un concepto que permite organizar estructuralmente las abstracciones y la ***herencia*** es una técnica de los lenguajes de programación que permite implementarla

# AUTO EVALUACIÓN/1

Comprendí los conceptos más importantes de la unidad 1.1 si puedo definir y dar ejemplos de:

- Análisis OO / Diseño OO
- Modelo de dominio de la aplicación
- Abstracción / Encapsulamiento
- Estado / Comportamiento / Identidad
- Tarjetas CRC
- Mensaje / Operación / Método
- Asociación / Agregación / Composición
- Generalización / Herencia / Polimorfismo

# AUTO EVALUACIÓN/2

Comprendí los conceptos más importantes de la unidad 1.1, si

- Comprendo la diferencia entre el análisis y diseño Estructurado y el OO
- Entiendo la diferencia entre análisis, diseño e implementación y que es lo que realizo en cada una de estas actividades
- Vinculo la etapa de análisis con la descripción del modelo de dominio de la aplicación
- Comprendo cual es el uso de las tarjetas CRC
- Entiendo cual es el objetivo de la abstracción y el encapsulamiento
- Entiendo la diferencia entre mensajes, operaciones y método
- Comprendo la diferencia entre asociación, agregación y composición
- Entiendo la diferencia entre generalización y herencia

Fin de la clase

UAI
Universidad Abierta Interamericana