

Introducción a Orientación a Objetos

¿Qué es un objeto?

- ✍ El mundo está lleno de objetos:
 - en la naturaleza
 - en entidades hechas por el hombre
 - y en los productos que usamos.
- ✍ Pueden ser clasificados, descritos, organizados, combinados, creados y manipulados.
- ✍ Es por ello que se utiliza una visión Orientada a Objeto para la creación de SW para computadora.

1

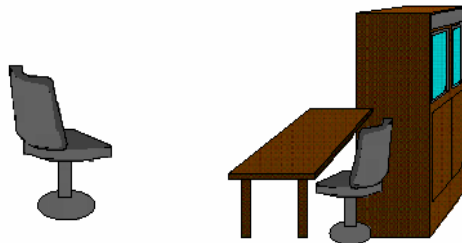
- Informalmente, un objeto representa una entidad del mundo real
- Entidades Físicas
 - (Ej.: Vehículo, Casa, Producto)
- Entidades Conceptuales
 - (Ej.: Proceso Químico, Transacción Bancaria)
- Entidades de Software
 - (Ej.: Lista Enlazada, Interfaz Gráfica)

2

¿Qué es un objeto?

✍ Ejemplo del concepto de Orientación a Objetos

- Un objeto del mundo real: una silla.
- ✍ La silla es un **miembro (instancia)** de una **clase** mucho más grande de objetos que llamaremos mobiliario.



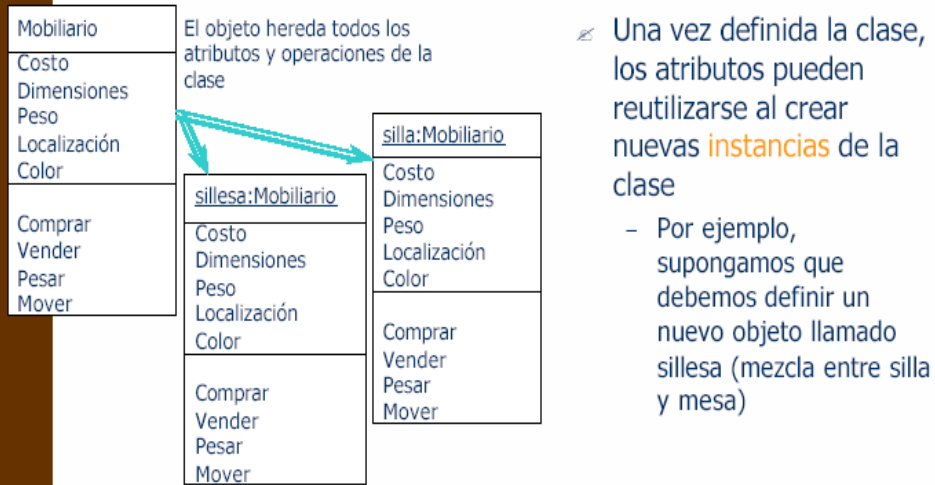
3

¿Qué es un objeto?

- ✍ Un conjunto de **atributos** genéricos pueden asociarse con cada objeto, en la clase mobiliario.
 - Por ejemplo, todo mueble tiene un costo, dimensiones, peso, localización y color, entre otros muchos posibles atributos.
 - Estos atributos también son aplicables a una mesa o silla, un sofá o un armario.
- ✍ Como silla es **miembro de la clase** mobiliario, silla **hereda** todos los atributos definidos para la clase.

4

¿Qué es un objeto?



5

¿Qué es el Análisis y Diseño OO?

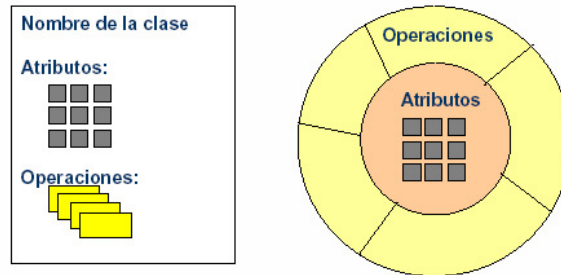
- En esencia es identificar el dominio del problema y su solución lógica dentro de la perspectiva de la Orientación a Objetos.
- Análisis Orientado a Objeto:** Identificar y definir los objetos (conceptos) dentro del dominio del problema.
- Diseño Orientado a Objetos:** Definir los objetos lógicos de Software (con atributos y métodos) que serán programados en un lenguaje de programación idóneo.
- Diferencia con el modelo estructurado:** El análisis y diseño estructurado son orientado a los procesos.

6

Introducción a Orientación a Objetos

Principios

- Las abstracciones de datos (atributos) que describen la clase están encerradas por una "muralla" de abstracciones procedimentales capaces de manipular los datos de alguna manera (**Encapsulamiento**).



7

Introducción a Orientación a Objetos

Principios

- Esto posibilita el **ocultamiento** de información y reduce el impacto asociado a los cambios.
- Como estos métodos manipulan un número limitado de atributos (alta **cohesión**) y como la comunicación sólo ocurre a través de los métodos que encierra la "muralla", la clase tiende a un bajo **acoplamiento** con otros elementos del sistema.

8

Introducción a Orientación a Objetos

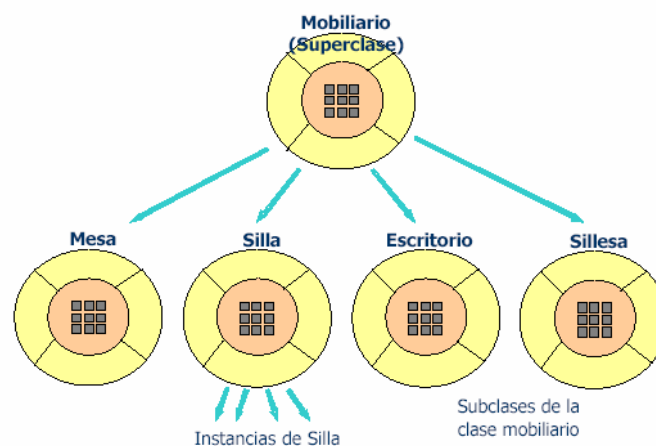
Jerarquía

- ✍ **Clase:** colección de objetos similares, los cuales heredan atributos y operaciones disponibles para la manipulación de éstos.
- ✍ **Superclase:** colección de clases.
- ✍ **Subclase:** instancia de una clase.
- ✍ Estas definiciones implican la existencia de una **jerarquía** de clases, en la cual los atributos y operaciones de la superclase son heredados por subclases que pueden añadir atributos "privados" y métodos.

9

Introducción a Orientación a Objetos

Jerarquía



10

Introducción a Orientación a Objetos

Atributos

- ✍ Los **atributos** están asociados a clases y objetos, describiéndolos de alguna manera.
 - Esta relación implica que un atributo puede tomar un valor definido por un **dominio** enumerado.
 - ✍ Una clase **auto** tiene atributo *color*. El dominio de valores de *color* es {blanco, negro, azul, rojo, amarillo, verde}.
 - En situaciones más complejas, el dominio puede ser un conjunto de clases.
 - ✍ La clase **auto** también tiene un atributo *motor* que abarca los siguientes valores de dominio: {valor 32 válvulas opción de lujo, valor 24 válvulas opción deportiva, valor 15 opción económica}.
 - ✍ Estas características se representan como asociaciones, no atributos.

11

Introducción a Orientación a Objetos

Operaciones, métodos o servicios

- ✍ Un objeto **encapsula** datos y algoritmos que procesan estos datos.
- ✍ Cada operación proporciona uno de los **comportamientos** del objeto.
- ✍ La operación se ejecuta en la medida en que se reciba un estímulo - **mensaje**.

12

Introducción a Orientación a Objetos

Ejercicio





- ✍ Describa a una bicicleta como clase definiendo dos operaciones y al menos tres atributos.
- ✍ Escriba el código java para la clase bicicleta sin especificar los métodos.



13

Introducción a Orientación a Objetos

Ejercicio: Posible Solución

Bicicleta
 vel_actual : int  cambio_actual : int
 cambiarVelocidad()  frenar()

```

Public class Bicicleta {

    private int vel_actual;
    private int cambio actual;

    public int cambiarVelocidad(int vel_nueva){}
    public int frenar(){ }

}
  
```

14

Introducción a Orientación a Objetos

Encapsulamiento, herencia y polimorfismo

- ☞ Una de las diferencias claves entre sistemas OO y sistemas convencionales es la **herencia**.
 - Una subclase Y **hereda** todos los atributos y operaciones de su superclase X
 - ☞ Todas las estructuras de datos y algoritmos originalmente diseñados e implementados para X están inmediatamente disponibles para Y, realizándose la reutilización en forma directa.
 - ☞ Cualquier cambio dentro de una superclase se hereda inmediatamente por todas las subclases que derivan de ella.
 - En cada nivel de la **jerarquía** de clases pueden añadirse nuevos atributos y operaciones a aquellos que han sido heredados de niveles superiores de la jerarquía.

15

Introducción a Orientación a Objetos

Encapsulamiento, herencia y polimorfismo

- ☞ A veces es tentador heredar algunos atributos y operaciones de una clase y otros de otra clase, ésta acción es conocida como **herencia múltiple**, lo cual complica la jerarquía de clases y puede crear problemas potenciales en el control de la configuración.

16

Introducción a Orientación a Objetos

Encapsulamiento, herencia y polimorfismo

- ✧ El polimorfismo es una característica que reduce en gran medida el esfuerzo necesario para extender un sistema.
- ✧ Considere una aplicación convencional que debe dibujar cuatro tipos diferentes de gráficos:
 - ✧ gráficos de líneas
 - ✧ gráficos de tortas
 - ✧ histogramas
 - ✧ diagramas de Kiviat.
- Idealmente una vez recogidos los datos para un gráfico particular, éste debe dibujarse a sí mismo.

17

Introducción a Orientación a Objetos

Encapsulamiento, herencia y polimorfismo

- ✧ En una aplicación convencional será necesario desarrollar módulos de dibujos para cada tipo de gráficos y añadir una lógica de control semejante a la que sigue:

```

case of tipo_grafico:
  If tipo_grafico = grafico_linea then DibujarLinea (datos);
  If tipo_grafico = grafico_torta then DibujarTortaLinea (datos);
  If tipo_grafico = grafico_histograma then DibujarHisto (datos);
  If tipo_grafico = grafico_kiviat then DibujarKiviat (datos);
end case
  
```

18

Introducción a Orientación a Objetos

Encapsulamiento, herencia y polimorfismo

✎ **Polimorfismo** permite que un número de operaciones diferentes tengan el mismo nombre, haciendo que cada objeto sea más independiente.

- La definición e implementación general del método reside en la superclase
- La implementación particular del método reside en la subclase
- La invocación es resuelta al momento de ejecución

19

Paradigmas de Programación

- Hay para todos los gustos
 - Estructurados (C, Pascal, Basic, etc.)
 - Funcionales (CAML)
 - Declarativos (Prolog)
 - Orientados a Objetos (C#, VB.NET, Smalltalk, Java)
 - Orientados a Aspectos
 - Híbridos (Lisp, Visual Basic)
 - Incomprensibles....
- Cada enfoque tiene sus ventajas y desventajas
- Cada uno es más apropiado para ciertas cosas

20

- ¿Por qué Orientación a Objetos (OO)?
 - Se parece más al mundo real
 - Permite representar modelos complejos
 - Muy apropiada para aplicaciones de negocios
 - Las empresas ahora sí aceptan la OO
 - Las nuevas plataformas de desarrollo la han adoptado (Java / .NET)

21

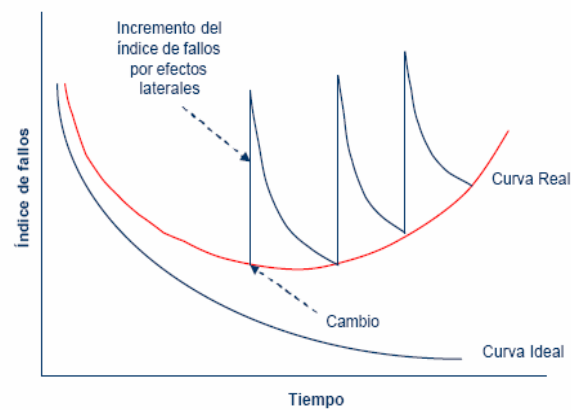
Problemas actuales en el desarrollo de SW

- ✍ El desarrollo de software es un negocio riesgoso.
- ✍ Muchas historias de fracaso [Larman]
 - 31 % Proyectos no se concluyen (estimativo)
 - 53 % Rebasa en un 200% el costo estimado (estimativo)
- ✍ Carencia de estándares.
- ✍ Dificultad en la estimación de costos.
- ✍ Poca reutilización de código.
- ✍ Proceso no definido.

22

Problemas actuales en el desarrollo de SW

☞ Curva típica de fallos de Software [Pressman].



23

Mitos del Software [Pressman]

Mitos de Gestión

- ☞ **Mito 1:** Tenemos un libro lleno de estándares y procedimientos para construir software. Mi gente ya tiene todo para hacer lo que tiene que hacer.
 - Saben que existe?, lo usan?, es completo?. Experiencia es también valiosa.
- ☞ **Mito 2:** Tenemos todo lo que necesitamos, HW y SW de última generación.
 - El desarrollo implica mucho más que herramientas.
- ☞ **Mito 3:** Si fallamos en la planificación, agregamos más programadores y asunto solucionado.
 - En general más gente agrega más caos no más eficiencia.

24

Mitos del Software [Pressman]

Mitos del cliente

- ✎ **Mito 1:** Una declaración general de los objetivos es suficiente para comenzar a escribir programas, los detalles se dejan para más adelante.
 - Una mala definición inicial es la principal causa de la pérdida de trabajo en SW.
- ✎ **Mito 2:** Los requisitos del proyecto cambian continuamente, pero estos pueden acomodarse fácil puesto que el software es flexible.
 - El impacto del cambio varía según en el momento que se introduzca.
- ✎ **Mito 3:** No es necesario involucrarse en el diseño del SW. Basta con dar las especificaciones y ver los resultados finales.

25

Mitos del Software [Pressman]

Mitos del desarrollador

- ✎ **Mito 1:** Una vez que escribimos el programa y hacemos que funcione nuestro trabajo ya está hecho.
 - -Cuanto más pronto se comience a escribir código, más se tardará en terminarlo.
- ✎ **Mito 2:** Hasta que no tengo el programa ejecutándose, realmente no tengo forma de comprobar su calidad.
 - Existen mecanismos efectivos para ser aplicados desde el principio del proyecto durante todas las fases.
- ✎ **Mito 3:** Lo único que se entrega al terminar el proyecto es el programa funcionando.
 - Un parte fundamental para la mantención es la documentación.

26

¿Porque es necesario un proceso formal?

- ✍ Tentación clásica: Ponerse a programar de inmediato.
- ✍ **Objetivo principal:** Disminuir el riesgo creando un sistema eficiente mediante un diseño formal.
- ✍ Es mas barato realizar cambios en las actividades de análisis y diseño que en la codificación.
- ✍ Reducción de complejidad, modelando sistemas y realizar abstracciones para detectar solo detalles fundamentales.
- ✍ Generación de modelos a bajo costo.
- ✍ Reutilización de código es una parte fundamental en la construcción de SW de alta calidad.

27

Procesos de desarrollo de SW.

- ✍ **Ciclo de vida:** Sucesión de etapas por las que atraviesa un producto software a lo largo de su desarrollo y existencia.
- ✍ Existen distintas formas o paradigmas de ciclo de vida:
 - Secuencial, cascada.
 - Orientado a prototipos
 - Evolutivo
 - ✍ Incremental
 - ✍ Espiral
 - ✍ Componente

28

Procesos de desarrollo de SW.

¿Como elegir un modelo?

✍ Algunos criterios:

- Complejidad del problema y solución.
- Frecuencia esperada en el cambio de requisitos por parte del cliente.
- Acceso al cliente por parte de los desarrolladores.
- Grado de exactitud en los requisitos.
- Tolerancia al riesgo.
- Presupuesto.

29