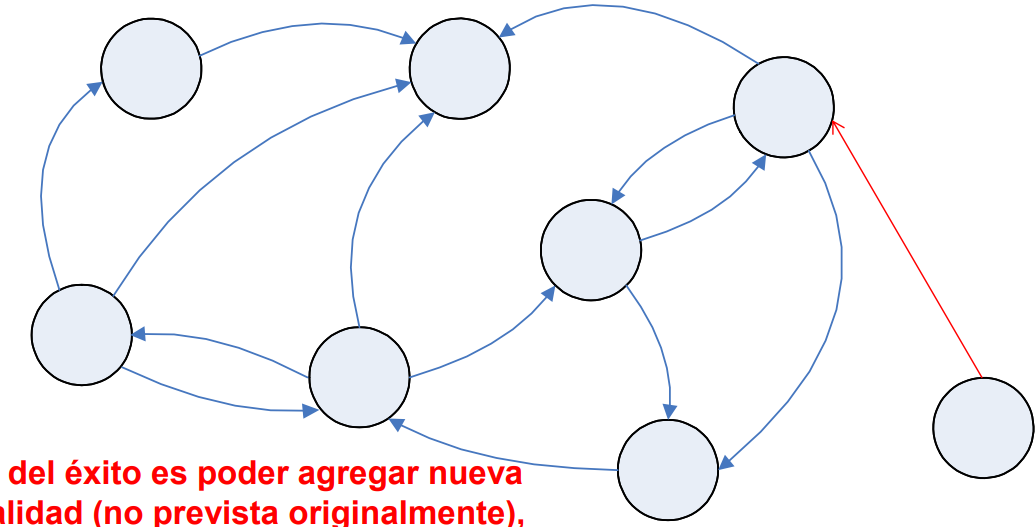
Orientacion a Objetos 1:

Teoria 1 – 29/8/24:

Software construido con objetos:

Es un conjunto de objetos que colaboran enviándose mensajes. Todo computo ocurre dentro de los objetos.



La clave del éxito es poder agregar nueva funcionalidad (no prevista originalmente), reemplazar objetos o modificar objetos y que el sistema “no se entere”, ni se rompa. E.g. integración Whatsapp y messenger Facebook

Los sistemas están compuestos (solamente) por un conjunto deobjetos que colaboran para llevar a cabo sus responsabilidades

Los objetos son responsables de:

* conocer sus propiedades
* conocer otros objetos (con los que colaboran)
* llevar a cabo ciertas acciones.

Aspectos de interés:

* No hay un objeto main.
* Cuando codificamos, describimos (programamos) clases
* Una jerarquía de clases no indica lo mismo que la jerarquía top-down
* Cuando se ejecuta el programa lo que tenemos son objetos que cooperan y que se crean dinámicamente durante la ejecución del programa
* Podemos pensar la interacción usuario/software de la misma manera
* Este mismo modelo nos permite entender (al menos en parte) otros modelos de computación: viendo a los objetos como proveedores de servicios, por ejemplo
* Este mismo modelo no asume objetos localizados en el mismo espacio de memoria (pueden estar distribuidos)

Impacto en como pensamos el software:

* Mientras que la estructura sintáctica es “lineal”, el programa en ejecución no lo es.

¿Qué es un objeto?

* Es una abstracción de una entidad del dominio del problema. Ejemplos: Persona, Producto, Cuenta Bancaria, Auto, Plan de Estudios….
* Puede representar también conceptos del espacio de la solución (estructuras de datos, tipos “básicos”, archivos, ventanas, iconos...)

Un objeto tiene:

* Identidad
  + Para distinguir un objeto de otro
* Conocimiento
  + En base a sus relaciones con otros objetos y su estado interno
* Comportamiento
  + Conjunto de mensajes que un objeto sabe responder

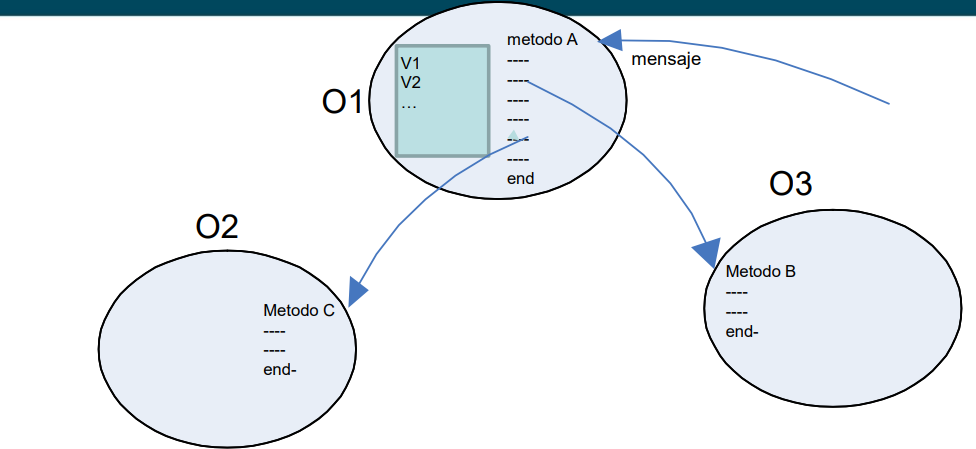
El estado interno:

* El estado interno de un objeto determina su conocimiento.
* El estado interno esta dado por:
  + Propiedades básicas (intrínsecas) del objeto. (dni, apellido, nombre)
  + Otros objetos con los cuales colabora para llevar a cabo sus responsabilidades.
* El estado interno se mantiene en las variables de instancia (v.i.) del objeto.
* Es privado del objeto. Ningún otro objeto puede accederlo. (¿Cuál es el impacto de esto?)

Variables de instancia:

En general las variables son REFERENCIAS (punteros) a otros objetos con los cuales el objeto colabora.

Algunas pueden ser atributos básicos



En el grafico O1 puede mandarle mensajes a O2 y O3 porque “los conoce”, o sea hay una variable en O1 que APUNTA a O2 y otra a O3 (o la misma variable que cambia de valor en diferentes momentos)

Comportamiento:

Un objeto se define en términos de su comportamiento.

El comportamiento indica qué sabe hacer el objeto. Cuáles son sus responsabilidades.

Se especifica a través del conjunto de mensajes que el objeto sabe responder: protocolo.



Implementación de comportamiento:

La realización de cada mensaje (es decir, la manera en que un objeto responde a un mensaje) se especifica a través de un método.

Cuando un objeto recibe un mensaje responde activando el método asociado.

El que envía el mensaje delega en el receptor la manera de resolverlo, que es privada del objeto



Envio de mensajes:

Para poder enviarle un mensaje a un objeto, hay que conocerlo.

Al enviarle un mensaje a un objeto, éste responde activando el método asociado a ese mensaje (siempre y cuando exista).

Como resultado del envío de un mensaje puede retornarse un objeto.

Lo que circula en los mensajes son objetos.

¿Cómo se especifica un mensaje?

* Nombre: correspondiente al protocolo del objeto receptor.
* Parámetros: información necesaria para resolver el mensaje.

Ejemplo:

cuenta.depositar(cantidad)

figura.dibujar()  
figuraGrande.rotar(45)

cuenta, figura, figuraGrande son variables que apuntan a un objeto que entiende el mensaje correspondiente

Métodos:

Que es un método?

* Es la contraparte funcional del mensaje.
* Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el cómo).

Un método puede realizar básicamente 3 cosas:

* Modificar el estado interno del objeto.
* Colaborar con otros objetos (enviándoles mensajes).
* Retornar y terminar.

Nunca hay script entrada y salida.

Programamos clases, no objetos.

Las clases son moldes

Hay librerías de clases y hay librerías de funciones

Como creamos objetos?

Comúnmente se utiliza la palabra new para instanciar nuevos objetos.

Los objetos se van creando bajo necesidad.

Ej: voy al banco y quiero abrir una cuenta, se crea un objeto cuenta, y quiero abrir una caja de ahorro, se crea el objeto de la caja de ahorro. Se conecta todo como un grafo que el banco conoce la cuenta, la cuenta la caja de ahorro, el banco conoce la caja de ahorro, etc.

Teoría – 5/9/24:

Relaciones objetosas:

Objetos que conocen a otros, identidad e igualdad, relaciones uno a muchos, delegación, polimorfismo y el rol de los tipos y las interfaces.

Los objetos son instancias de clase

Las clases son estructuras

Cuando se hace el new se mira la clase, se reserva espacio y se le pone la cantidad de variables q dice la clase y se ponen los punteros para los objetos

El constructor se asegura de que el objeto esté listo para ser usado

Cuando una clase se instancia tiene las variables de instancia con un puntero a un objeto. Estas variables de instancia se mantienen hasta que yo lo cambie o el objeto se muera.

Un objeto conoce a otro porque:

* Es su responsabilidad mantener a ese otro objeto en el sistema
* Necesita delegarle trabajo (enviarle mensajes)

Un objeto conoce a otro cuando:

* Tiene una referencia en una variable de instancia
* Le llega una referencia como parámetro
* Lo crea
* Lo obtiene enviando mensajes a otros que conoce.

THIS:

* This.mensaje() el objeto se manda un mensaje a si mismo
* Locomotion.move(this) el objeto se manda como parámetro
* This.variable Cuando queremos quitar la ambigüedad entre variables. Sirve como para decir acá me quiero referir a la variable de instancia.

Que hace el objeto cuando recibe un mensaje? Lo busca en su clase a ver si hay un método para ese mensaje.

Identidad / el operador “==”:

* Las variables son punteros a objetos
* Mas e una variable pueden apuntar a un mismo objeto
* Para sabre si dos variables apuntan al mismo objeto utilizo “==”. Checkeo de identidad para ver si 2 punteros apuntan al mismo objeto
* == es un operador, no puede redefinirse.

Si yo quiero saber si un auto es igual a otro auto, uso el “equals”.

Marca.equals(otroAuto.getMarca()) checkea si son “parecidos” y por ejemplo marca toyota, son lo mismo pero son diferentes objetos.

Se podría usar el “==” para que el objeto toyota sea siempre el mismo y tenga herencia.

Chequeo de tipos:

* Java es un lenguaje fuertemente tipado:
  + Debemos indicar el tipo de todas las variables
  + El compilador chequea la correctitud de nuestro programa respecto a tipos.
* Se asegura de que no enviamos mensajes a objetos que no los entienden.
* Cuando declaramos el tipo de una variable, el compilador controla que solo “enviemos a esa variable” mensajes acordes a ese tipo.
* Cuando asignamos un objeto a una variable, el compilador controla que su clase sea “compatible” con el tipo de la variable.

Tipos en lenguajes OO:

* Un tipo es un conjunto de firmas de métodos
* Firmas de métodos son nombre, orden y tipo de los organimentos.
* Cada clase en java define explícitamente un tipo.
* Algunos lenguajes diferencian entre tipos primitivos y tipos de referencias (objetos)
* Asignar un objeot a una variable, no afecta al objeto.

Robot r

Object o = r (acá o es una variable y r también, cada una tiene su puntero. Pero con el igual, o apunta a lo mismo que r)

Entonces al hacer o == r, me da True.

Interfaces:

* Es un tipo.
* Che en esta variable tengo algo de este tipo y no se de que clase es, ni tampoco me importa.
* Solo son firmas de métodos.

Cuanco una clase implementa una interfaz siempre antes chequea que existan los métodos que llamo así no revienta el programa

Teoría – 12/9/24:

El super empieza a buscar desde arriba de todo, da igual donde lo llame

Por ejmplo: super.actionOne() lo ejecuto en “C”. Cuando se ejecute C y llega a la línea del super, lo va a buscar en el mismo, si no lo encuentra, lo busco arriba, en “B”, si B lo tiene, se ejecuta, pero si b lo tiene con un super, lo busca arriba también, en “A”.

Teoría – 19/9/24:

UML:

Tipos comunes de interfaces:

* Java.util.List: permiten duplicados
* Java.util.Set: no permiten duplicados
* Java.util.Map: asocia objetos que actúan como claves a otros que actúan como valores.
* Java.util.Queue: cola.