**REPASO PARA PARCIAL METODOLOGÍA**

Me ayuda a ***organizarme*** en el desarrollo, operación y mantenimiento de software, permite desarrollar un ***sw de calidad*** (entre otros aspectos, construir lo que realmente quiere el cliente), para ello → ***provee métodos y herramientas***.

**DISCIPLINAS**

- **Técnica** → Nos permite construir el producto (SW), nos dice como construirlo. (en esta materia)
- **Gestión** → Nos ayuda durante el proceso de construcción, nos brinda guías. (***Estimar, Planificar el Proyecto*** → lo que haremos, cuándo, cómo y quién, ***Monitoreo y Control de Proyectos*** → compara plan con avance real)
- **Soporte** → Nos da herramientas para construir SW de calidad (dde dif. Aspectos del sistema, con objetivo final → construir el SW que quiere el cliente.) ***Gestión de Configuración de SW*** → crear mecanismos que nos ayuden a controlar el software a lo largo de su ciclo de vida, ***Aseguramiento de Calidad*** → actividad de protección que se aplica a lo largo de todo el proceso de construcción. ***Revisar técnicamente el software*** → Tiene como objetivo detectar tempranamente errores que se cometen al desarrollar.

**DISCIPLINA TÉCNICA (Qué Hacer P/Desarrollar El SW, que actividades)**

**ETAPAS DEL CICLO DE VIDA:**

**1) CAPTURA DE REQUERIMIENTOS** → Entender del dominio y lo que quiere el cliente (charla), abstraer la información y quedarnos con lo necesario para el desarrollo. Escritura en lenguaje natural.
**2) ANÁLISIS** → Paso del lenguaje natural a uno + específ. De desarrolladores.
***1) y 2) Qué tengo que solucionar...***
**3) DISEÑO** → Bosquejar una solución para mi problema. ***Como voy a solucionar….***
**4) IMPLEMENTACIÓN** → Escribir el código de lo diseñado.
**5) TESTING** → Prueba de lo codificado, hay muchos tipos de test, ejemplo dde el pto de vista funcional, test estrés etc. ***Verificación***(Sin bugs, funciona OK) ***y Validación***(cumple con lo requerido por el cliente) ***V&V*** → *Lo hace el equipo de QUA , gralmente externo, busca encontrar errores* → *DESTRUCTIVO.*
**6) DESPLIEGUE (deployable)** → Cuando lo instalo dentro del cliente, llevo todos los artefactos que necesito para que el sistema comience a funcionar.

**CAPTURA DE REQUERIMIENTOS**
Condición o capacidad, Informada por el cliente, que debe cumplir el sistema, permite delimitar el sistema (que cosas SI se van a implementar y cuales NO)

**REQUERIMIENTOS FUNCIONALES (*como la opción en un menú*)** → **Solo estos en Captura de Req**
**REQUERIMIENTOS NO FUNCIONALES** (restricciones, atributos de calidad, req no comportamentales etc.) → Son ***restricciones que atraviesan a uno o varios req funcionales*** (Ej si uso BD relacional o BD orientada a objetos, etc. ) Se trabajan en la etapa de diseño.

**TÉCNICAS DE ESPECIFICACIÓN DE REQUERIMIENTOS**

Primeras estrategias → Entrevistas, cuestionarios.
Actualidad → *Casos de Uso* (UML) , *Historias de Usuario (User Sories)* → Met Ágiles

**DESARROLLO DE SOFTWARE (4P: Proyecto, Proceso, Personal, Producto)**

*** PROYECTO (**la ejecución del Proceso**)** →
*** se adapta-> PROCESO(**Las actividades a realizar**)**, me da un marco de trabajo como referencia, que guiará el proyecto. ***El Ciclo de vida del proceso de desarrollo de SW*** → etapas y su orden
Estrategias → Antes → Secuencial(c/etapa completa). Después → Iterativo(dividido en subsistemas)
**MODELOS** → Ciclo de vida en cascada (la etapa del sist. completo y paso a la px)
→ Ciclo de vida Incremental (toma la estrat. Iterativo). Construye peq. Porcion del sist. Final que se puede entregar al cliente.

→ Ciclo de vida Iterativo (Divido en subproblemas, aplico cascada)
→ **Ciclo de vida Iterativo Incremental** (peq ciclos de vida en cascada +incremento)

*** se incorpora → PERSONAL(**los perfiles necesarios**),** personas asumen difer. Roles en equipo de desarrollo (Analista en Sistemas, Diseñadores, Programadores, Arquitectos, Analistas de pruebas, Lideres de proyectos, Revisores técnicos).
*** Se obtiene → PRODUCTO (**A demás del ejecutable, todos los artefactos que voy a generar para el sistema, deben ser consistentes con producto final**)**
Ayudado por **HERRAMIENTAS**

## METODOLOGIA DE DESARROLLO DE SOFTWARE

Hay diferentes formas de organizarnos, la metodología me dice cómo tenemos que organizarnos, cuál es el ciclo de vida a usar para desarrollar ese sistema, y cuáles son los artefactos, los diagramas y documentos a generar en c/etapa. Dependiendo de las características de mi sistema, voy a ver qué metodología se adapta mejor al sistema que tengo que desarrollar.

**Hay muchas, TIPOS:**
primeras → **Orientado a dato o función** → **p**aradigma procedural, datos y funciones
luego → **Orientado a Objetos** → encapsula datos y funciones.
Como mejora surgió → **Mét. Agiles** → me ayuda a trabajar con requerimientos cambiantes.
**Métodos formales** → donde la vida humana corre peligro, control aéreo, de trenes, centrales nucleares , NO puede haber errores en los requerimientos, surgen para VALIDAR formalmente esos requerim. , tienen una base MTM, Algebraica o Lógica que me permite asegurar que el requerimiento fue bien capturado.

**En resumen:**

**Un PROCESO de desarrollo de software** → define la *metodologia* de desarrollo de software a utilizar y los *roles* necesarios para el desarrollo
**Una METODOLOGÍA de desarrollo de software** → define el *ciclo de vida* que se utilizará y los *artefactos* que se deben generar en cada una de las etapas del ciclo de vida. ***SIEMPRE NECESITAMOS SEGUIR UNA.*** **+da Roles**
**El CICLO DE VIDA de un sistema** → define el orden en el cuál las *etapas* del ciclo de vida se desarrollarán
**Las ETAPAS del ciclo de vida** → definen las *actividades* necesarias para desarrollar un sistema

## *SCRUM (hoy día → es una metodología)*

**Planificar → Ejecutar → Reflexionar**

- Tiene eventos, roles y artefactos.
- Es ***Autodirigido*** y ***Autoorganizado*** → A nivel del equipo de desarrollo (DT)
- Se crea una iteracción y se comienza (el cliente no puede agregar nuevo trabajo a la misma)
- Tiene reunión diaria del DT para sincronizar actividades
- Al final de cada iteracción → hay ***DEMO***
- Cada iteracción, está guiada por el cliente.

**PILARES FUNDAMENTALES**

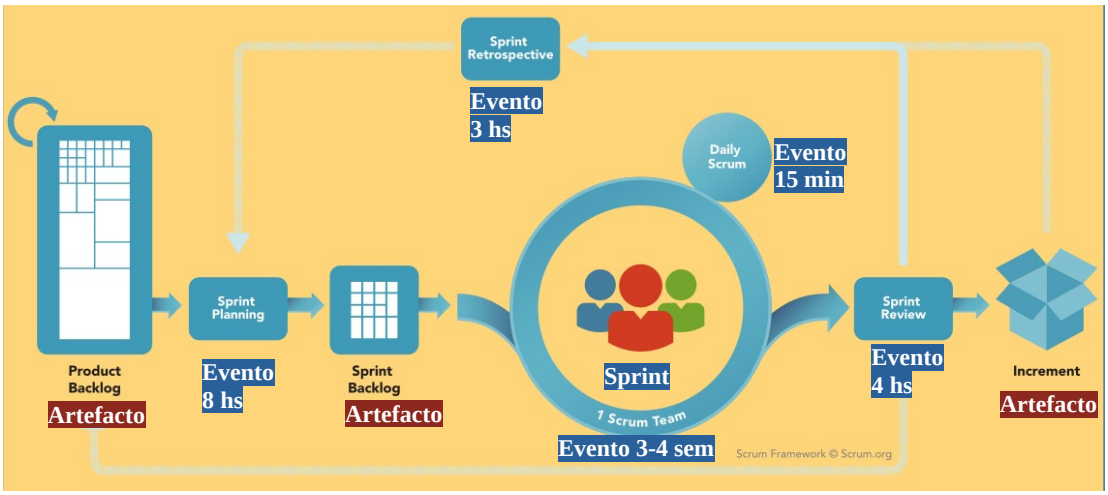
**Transparencia** → hacia el cliente, éste está todo el tiempo consciente del estado del proyecto.
**Inspección** → Mirar los artefactos que construimos y el proceso (como nos comunicamos, como estamos trab. Etc.)
**Adaptación** → Mejorar tanto artefactos, como proceso. Para ello tiene 4 eventos: Sprint planning, Daily Scrum, Sprint review, Sprint retrospective.

Muy ***centrado en el DT*** → Falla por falta de compromiso del equipo, debe compartir **VALORES** → ***Coraje*** (hacer lo correcto por más de que sea difícil) – ***Foco*** (centrado en el objetivo de la iteración) – ***Compromiso*** – ***Respeto*** – ***Apertura*** (aceptar cambios a partir del cliente).
**ROLES**

**Scrum Team**, compuesto por:

- **Product Owner (PO)** → Única persona. Responsable del proyecto del lado del cliente, sabe cuál es el sistema a implementar. Debe: *Expresar, priorizar y ayudar a entender los requerimientos. Establece objetivo de c/iteracción.*
- **Development Team (DT)** → Equipo de desarrollo. 7+-2 personas. Se autogobierna. No hay etiquetas, todos tienen que estar disponibles para codear. NO hay subequipos que se encarguen de algo en particular, todo lo hace el DT.
- **Scrum Master** (SM-"La policia de Scrum") → Sabe bien la metodología y nos ayuda a cumplir con todos los *valores, prácticas y reglas*, (pasos, eventos, artefactos…) NO es el jefe del equipo.

**CICLO DE VIDA → FRAMEWORK**
**iteracción = sprint**
**Tiene: a) Los artefactos a generar b) Los eventos**




**Artefactos:**

- **Product Backlog** → En primer reunión con el PO (x el cliente), se arma una lista ordenada (sin mucho detalle) de ***todos los requerimientos*** que tenemos que implementar en nuestro sistema. Todos los Product Backlog Item (PBIs ) del producto. Es responsabilidad del PO. Es un documento que vive durante todo el ciclo de desarrollo y va evolucionando.
- **Sprint Backlog** → surge a partir de Sprint plannig. Tenemos + detallados los requerimientos a implementar en el px sprint. Subconjunto de PBIs que se seleccionaron para el px sprint, y responden a un objetivo (goal) del sprint. Una vez que comenzó el sprint → **SOLO** puede ser modificado por el DT. (agregar detalles, no + trab.). Se usa Scrum Board para la organización (JIRA)
- **Incremento** → Suma de todos los PBIs que se completaron durante el sprint + los sprint anteriores. Una vez que termina debe estar en Done. Debe estar usable.

**Eventos:**

- **Sprint Planing** → Reunión SM, DT y PO, donde el PO ***prioriza los items*** del product backlog y vemos cuantos de estos se hacen en el px sprint. Se realizan todas las preguntas necesarias para obtener + detalle. El ***DT*** analiza cómo lo va a hacer y su estimación. Nos vamos con el objetivo del sprint (Sprint Goal) y su Sprint Backlog. Es time-box (max 8 horas p/sprint 1 mes)
- **Sprint** → donde realizamos las distintas partes en cascada de nuestro ciclo de vida original. Dura entre 3 y 4 semanas. Es time-box (a tiempo fijo). Muy raramente se cancela.
- **Daily Scrum** → ***Reunión diaria*** del DT, para sincronizar actividades y crear plan para px 24 hs. Es time-box (15 min) c/miembro responde: qué hice? Que voy a hacer? Estoy trabajo con algo? SM→ Se asegura que TODOS y SOLO los miembros del DT participen.
- **Sprint review** → ***Demo*** con el cliente. Se enfoca en el ***producto***. Obtenemos Feedback. Participa el Scrum Team completo + stakeholders (otros miembros de la empresa que saben del SW). Time-box (4 horas p/sprimt 1 mes). Tiene mucho protocolo.

- **Sprint retrospective → *Reflexionar a partir de lo que hicimos*** en el último sprint. Se enfoca en el ***proceso***. *Qué hicimos bien, que no?* Como podemos comunicarnos mejor. *Qué podemos cambiar?*. Participa el Scrum team pero esta enfocado en el DT. Enfocarnos en Como hacer mejoras. Time box(3 hs p/sprint 1 mes)

**Cuando termina el sprint → tengo INCREMENTO (Artefacto que se está generando → Software)**

**CRITERIO DE DONE (Definition of Done DoD) →A nivel de producto. UN DoD por Proyecto**

Ponerse de *acuerdo con el cliente*, que significa que el trabajo esté terminado, y con el DT.
*En general en una empresa:*
implementado, testeado, comentado, correctamente identado, comiteado, integrado en el branch que corresponda, coach review, buenas prácticas.
Done = Releasable (listo para empezar a ejecutar)
Hace lo que el cliente quiere que haga. Cuestiones de calidad. Tanto las US como las tareas deben cumplir con el DoD.

# PBIs EN SCRUM

SCRUM *NO dice nada de cómo escribir PBIs*, para ello hay varias *técnicas: User Stories (+usada) , Casos de uso*.
**Warm-UP** → sprint muy corto, para construir el Product Backlog y el prototipo.

**USER STORIES (US) → Es una "promesa" de conversación con el cliente**

Describen QUE es lo que se debe implementar, NO el como..
**Funcionamiento → 3C →**

- **Card** → tarjeta → Como **ROL** quiero **OBJETIVO** para **BENEFICIO**
- **Confirmation** → del otro lado de la tarjeta → Criterios de Aceptación (detalles, condiciones de satisfacción sobre el comportamiento del sistema, por parte del cliente. Para c/ US)
  ***Formas de Escribir los Criterios de Aceptación*** → Verificar/Testear/Chequear , Se debe poder…, Demostrar que…, Dado COND cuando ACCION entonces RESULT
- **Conversation** → Los detalles de la conversación se conversan en una reunión e/ DT, PO, STKholders

**Nivel de Detalle → Las US se pueden escribir con distinto nivel de detalle**

**Epica** → US con muy poco nivel de detalle, NO tenemos idea cuánto tiempo nos va a llevar.
**Feature** → US con más detalle, las podemos estimar pero son + grandes como p/ llevarlo a un sprint
**Sprint ready / Sprintable** → US con mucho nivel de detalle, se puede llevar a un sprint.
**Task / Tareas** → Las tareas que tenemos que hacer para una US Sprintable. En gral duran horas. Ya van al COMO.

**Escritura de Buenas US → INVEST** → Idependiente(p/ que no se bloquee c/otra), Negociable (xq es una promesa), Valuable(Tiene que tener un valor de negocio, beneficio), Estimable, Small(q se pueda pasar a done en 3-4 días), Testeable(Como parte de los criterios de Done).
**Refinar US → Resultado debe ser >= 2 US**

**USER STORIES MAPPING (USM) → Para ver todas las US en Contexto, ayuda a organizar y priorizarlas**

**1er nivel** → Roles de usuario
**2do nivel** → USER ACTIVITIES (Actividades ppales que c/usuario va a desarrollar en el sistema)
**3er nivel** → Back Bone (Tareas del Usuario) → en gral las Epicas.
**Último** → US feature o sprintables. Estas se pueden agrupar en ***Release*** según decida el cliente. **Release <> Sprint**
**MVP → Mínimo producto Viable** , qué es lo mínimo que necesita el cliente. → US definidas en el 1er release.

# ULM (Lenguaje de Modelamiento Unificado)

Provee un conjunto de diagramas visuales con el que comunico ciertas decisiones del desarrollo
**Tipos:**
**Diagramas Estructurales (estáticos) →**

- Diagrama de clases (representa conceptos a modelar, componentes lógicas del sist),
- Diagrama de objetos (modelar un instante particular de la ejecución del sistema, los objetos y sus relaciones), Diagrama de componente (para ver la topología del hardware, usado p/ sist. distribuidos),
- Diagrama deployment (Modela la implementación)

**Diagramas Comportamentales (dinámicos) →**

- Diagrama de casos de uso (modela la funcionalidad del sistema, límites y actores),
- Diagrama de iteración(objetos participantes en la ejecución de una funcionalidad, envío de mjes e/esos objetos):
  - Diagrama de secuencia (énfasis en el orden de envío mje)
  - Diagrama de colaboración (énfasis en los objetos q partic)

  Solo se dibujan distinto
- Diagrama de actividades, (actividades que se deben realizar p/ resolver una problem. en particular, se usa como complemento del diagr. De casos de uso)
- Diagrama de transición de estados. (modela diferentes estados de los objetos q partic.)

## CASOS DE USO (UC)

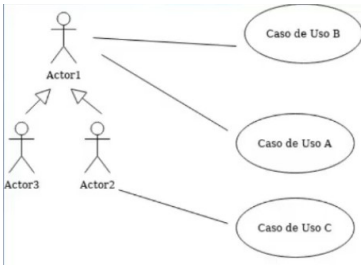
Para Captura de requerimientos. Permite expresar la funcionalidad del sistema, que información se recibe del exterior, que se genera y qué cosas debe hacer si sist. para desarrollar esa funcionalidad, me permite ver los límites del sistema.

***Elementos:***

- **El caso de uso →** representa una funcionalidad del sistema. Nombre en infinitivo.
  Verlo como *OPCIONES DEL MENÚ*.

- **El Actor →** representa el rol que juega la persona que interactúa con el sistema(el que ***TIENE*** la información, no el que la usa). Nombre de ese Rol.
  Una persona puede cumplir <> roles. Puede ser: humano, hardware, otro sistema
  *Solo 1 Actor primario → por c/ UC (**el culpable de que el sistema se active**)*
  *>= 0 Actor secundario x/c UC (brinda o recibe información del sistema Ej: vendedora que cobra por comis)*
  *Actor reloj → activación del sist. En un tiempo particular.*


***En diferentes momentos, un UC puede tener diferentes actores primarios.(Ej profesor, estudiante, VER nota) Una funcionalidad puede ser activada por varios roles. → Herencia de Actores***




- **Activación →** línea que une actor con Caso de uso.



- **Relaciones e/casos de uso →** Inclusión, Extensión, Generalización. (Para estructurar y organizar más claramente el diagrama de UC)

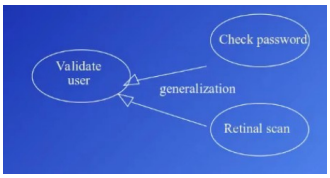
## Template de documentación para UC

**qué cosas va a hacer la funcionalidad, NO como, NO poner formulario, Base de datos, etc.**

- **Nombre del caso de uso:** Nombre representativo de dicha funcionalidad (*verbo en infinitivo*)
- **Descripción breve:** Descripción en un párrafo de lo que hace ese caso de uso
- **Actor primario/secundario(s)**: Actores involucrados en el CU
- **Trigger**: Situación por la que el caso de uso se activa. ***"El caso de uso comienza cuando el actor quiere..."***
- **Curso básico**: Flujo de ejecución ideal. 1° → trigger, luego oraciones cortas Sujeto, verbo,..Ej El sistema solicita datos personales, El socio ingresa su nombre, apellido… último → El CU finaliza
- **Cursos alternativos**: Desviaciones del flujo de ejecución ideal.
  Cuando hay un Verifica o un Si condicion, puede haber 2 o más respuestas.
  Nombre representativo, enumero los pasos del curso básico, y el el paso 6 donde hay un verifica, sumo subpasos 6.1, 6.2 ir al paso 7
- **Precondición**: Algo que debe ser verdadero (chequeado por el caso de uso) antes de desarrollar su funcionalidad
- **Postcondición**: Algo que debe ser verdadero (chequeado por el caso de uso) después de desarrollar su funcionalidad
- **Suposiciones**: Verificaciones que se suponen ya están comprobadas (el caso de uso no tiene que chequearla)
- **Casos de uso que extiende**: Casos de uso a los que incorpora nueva funcionalidad en situaciones particulares
- **Casos de uso incluidos:** Casos de uso que necesita para completar su funcionalidad
- **Finalización del caso de uso**: Situaciones por las cuales un caso de uso termina (se desarrolló correctamente, hubo algún alternativo que terminó el desarrollo del caso de uso, cancelación)

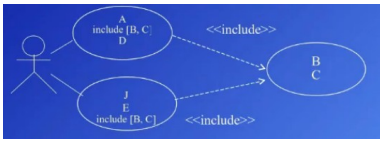
***Ejemplo → video clase 20/04 min 1:27***

## Relaciones entre CU

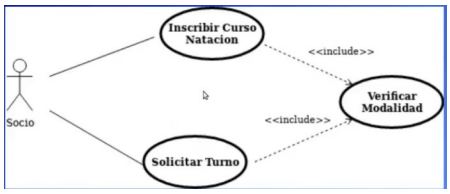



**Generalización**(no muy aceptada) → un cu padre hereda su comportamiento a sus hijos.
Ambos CU validan pero de diferente manera. Ej <> validaciones

**Inclusión** → (para no duplicar código)



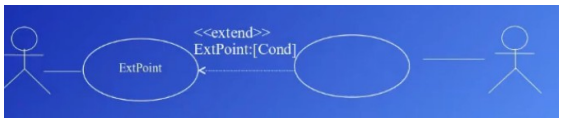

CU incluido tiene código del original y SOLO es activado por él (no por un actor)
Es explícita: ***[INCLUDE] Verificar Modalidad***
El CU incluido *DEBE estar incluido en + de 1 CU*.
***Ejemplo → video clase 23/04 min 32:25***

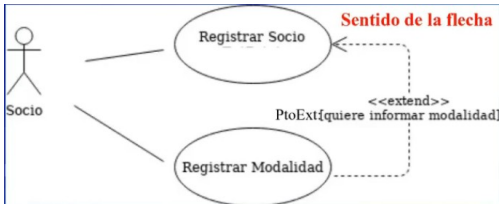



**Curso básico:**
1.El caso de uso comienza cuando un socio quiere inscribirse en un curso de natación.
2. [INCLUDE] Verificar modalidad. EXPLICITO
3.El sistema lista los cursos disponibles. Por cada curso se detallan los horarios de asistencia, el profesor que lo dicta y la fecha de comienzo del curso.
4.El socio selecciona uno de los cursos.
5.El sistema verifica que el curso todavía no haya comenzado.
6.El sistema controla la cantidad de inscriptos en el curso.
7.El sistema verifica que haya menos de 15 inscriptos
8.El sistema registra la inscripción al curso.
9.El caso de uso finaliza.
**Caso de uso incluido/que incluye: Verificar modalidad**

*Ej CU Loguear → incluido en todos los CU*
*ó Si quiero hacer algo → viene el CU Loguear que extiende, cuando no esté logeado* ***(mejor opción)***

**Extensión** → (para las desviaciones del curso normal)
Se da cuando el CU tiene un problema y otro CU se la soluciona.
Está implicito.
Es activado por un actor. Cuando el CU original es activado ***según una CONDICION***, el CU extends se activa.
**Un Cu que extiende DEBE también ser activado por un actor.**
***Ejemplo → video clase 23/04 min 55:00***
*"..puede informarlo en cualquier momento.."* → funcionalidad que puedo hacerla en cualquier momento




← Cond

**Curso básico:**
1. El caso de uso comienza cuando un socio quiere registrarse en el sistema.
2. El sistema solicita los datos personales.
3. El socio ingresa su nombre, apellido, email, dirección y edad.
4. El sistema solicita un identificador único.
5. El socio ingresa su número de documento.
6. El sistema verifica el identificador ingresado.
8. PtoExt IMPLICITO
9. El Sistema guarda la información del nuevo socio
10. El Caso de Uso finaliza

**Curso básico:**
1. El caso de uso comienza cuando un socio desea informar la modalidad de asistencia a la pileta
2. El sistema presenta las modalidades de asistencia a la pileta disponibles:
   *Curso de natación
   *Pileta libre
3. El socio selecciona una modalidad de asistencia.
4. El sistema actualiza la nueva modalidad.
5. El caso de uso finaliza.
Casos de uso extendidos:
Extiende el caso de uso Registrar socio. Cuando el socio quiere informar modalidad en el punto de extensión Ptoext

**PRECONDICIÓN/ POSTCONDICION** → Algo que debe ser V (el Cu SI la verifica )para que el CU pueda desarrollarse (ej p/ extpraer dinero, tiene que haber en cta).

**SUPOSICIÓN** → Verificaciones que se suponen ya están comprobadas (el CU NO la verifica)
***Caso particular*** → el login, ya que el diagrama sería con demasiadas flechas.

**Otro Caso de uso particular** → ABM → si son acepciones a la BD (insert, update, delete) se ponen en *un solo CU*, si hay que hacer otros pasos (chequeos, etc.), se hacen *separados*.

## DIAGRAMA DE CLASES

**Se usan para:** explorar conceptos del dominio, analizar requerimientos, mostrar diseño detallado del Soft. Or. Objetos
**Contiene: Clases, Interfaces y relaciones.**

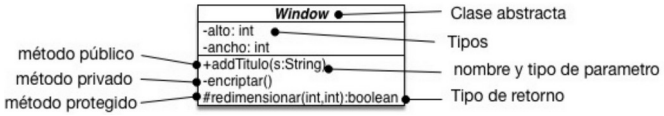
**CLASES** → descripción de un conjunto de objetos con iguales atributos, operaciones, semántica. (molde p' <> instancias).

Nombre → Sustantivo. 1ra letra May- CamelCase

Atributos → propiedades 1ra min luego CamelCase

Operaciones. Nombres casi siempre Verbos. 1ra en min luego CamelCase


**Accesibilidad de atributos y operaciones:**
**Public +** **Private -** **Protected #**
Si no ponés nada → package o frendy
**Clase abstracta** → Cursiva en su N*ombre*
**Tipo de atributo** → nombre:tipo
**Parámetros** → nombre:tipo o solo tipo, separados por coma
**Retornos de los métodos** → metodo ()**:tipo**
**Atributo estático** → nombre subrayado

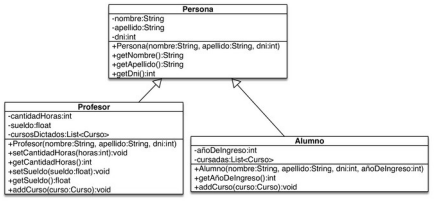



## RELACIONES ENTRE CLASES

Colaboración entre objetos. Dos clases relacionadas A y B, en tiempo de ejecuc. se envían algún tipo de mje ó se crea un tipo de la otra clase, ó tiene un atributo del tipo de B ó se recibe un mje con un objeto de B como parámetro, A es superclase de B ó A implementa B.

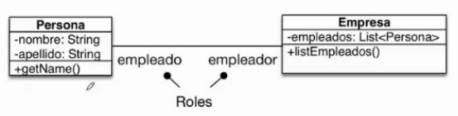
**Tipos**:

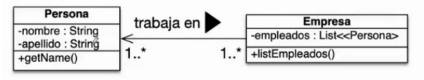
1. **Generalización (Herencia)**
   La subclase "es un tipo de" la superclase.
   Atributos, operaciones y relaciones comunes **se muestran en la superclass**.



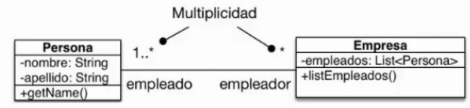

2. **Asociación → Relación fuerte**
   "Tiene…", "es de..","conoce.." → Asociación
   Es **Estructural** → A tiene un atributo del tipo B o colección.
   Modela conexiones del tipo: ***tiene, es de, conoce*** ——
   *Clases que son **pares**.*
   ***Adornos(solo para asoc) → tipos:***


   a. **Nombre/Rol** *(Obligatorio poner alguna de las 2*
   b. **Navegabilidad (**NO es Obligatorio, p/ indicar restricciones, indica que una clase conoce y le puede mandar mjes a la otra clase, pero no viceversa, se agrega flecha**)**

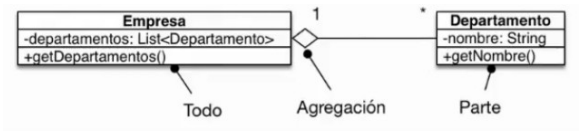


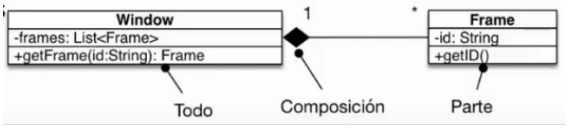

   c. **Multiplicidad (**Obligatorio. cuántos elementos de una instancia se relacionan con otra ej 6..* (6 o más obj)



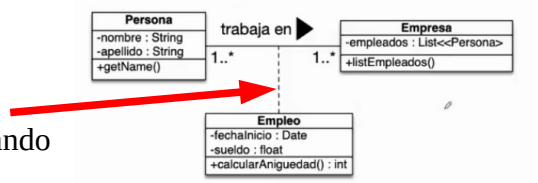
3. **<u>Agregación</u>** → tipo especial de relación de asociación, la *especializa*. Puede incluir adornos.→ ***Multiplicidad Oblig***
   Una clase juega el rol del ***todo***, y la/s otra de la ***parte***.
   (semánticamente, según lo exprese el cliente)
   ejemplo: "Está formada por…","se estructura por..." → Agregación



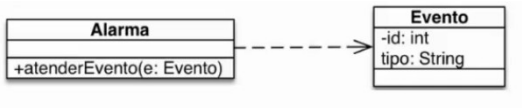

4. **<u>Composición</u>** → tipo especial de relación de agregación, la especializa.
   Una clase está ***compuesta*** por otras. Relaciona tiempo de vida del todo, con el tiempo de vida de las partes. (Preguntarnos → Cuando se destruye el todo, se destruyen las partes? O tiene sentido que sigan existiendo?)
   Puede incluir adornos.→ ***Multiplicidad Oblig***
   *semántica → "está compuesto por"*




5. **<u>Clase de asociación</u>** → Surge cuando tenemos un **N a N (muchos a muchos)**
   Permite crear una nueva clase que guarda la info de la relación.
   ***No Siempre*** que tengo N:N modelo esta clase de asociación, solo cuando necesito la información generada en la relación.

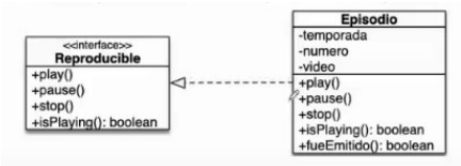



6. **<u>Dependencia</u>** → La relación mas débil que define UML.
   Indica que una clase usa a otra clase. Conexiones del tipo "usa"
   A depende de B cuando:
   1. Una *operación* de A define una variable del tipo B.
   2. Clase A envía mje a clase B
   3. Clase A crea un objeto de clase B
   4. Clase A recibe un mje con un objeto de la clase B como parámetro.
   5. Una Clase A importa a otra clase B

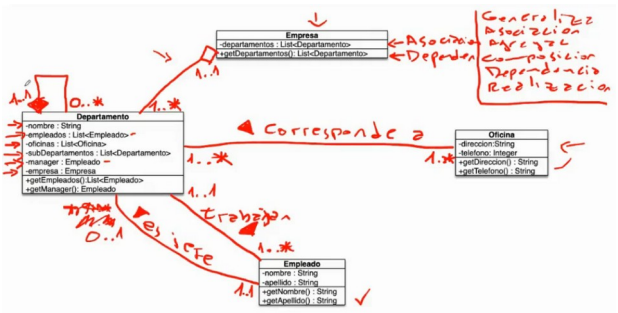



**Si tengo una relación fuerte y una débil, muestro solo la fuerte. Ej asociación y dependencia, muestro asoc.)**

7. **<u>Realización</u>** → Lo mismo que relación de herencia pero para interfaces
   conexiones del tipo "implementa.."




**Ejemplo práctico:**



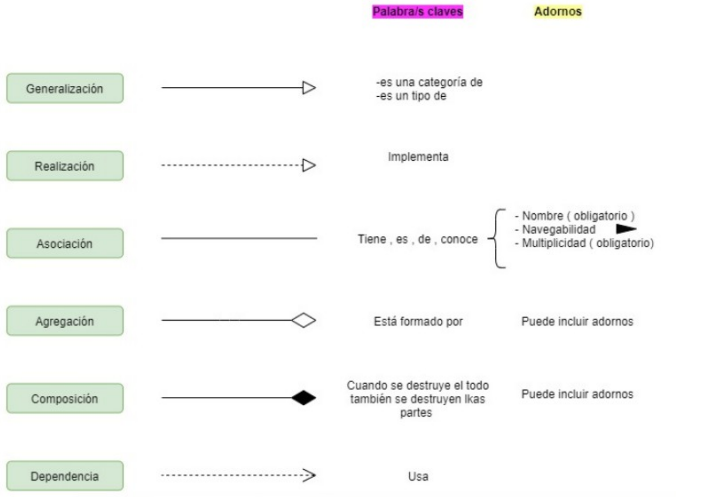
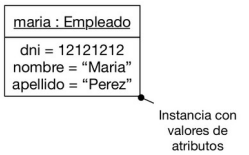
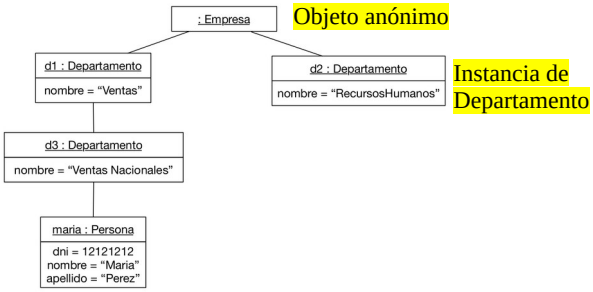



**Diagrama de objetos** → Modelan instancias



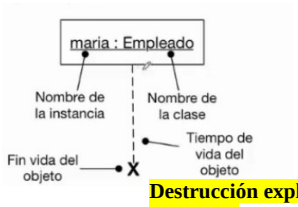

# DIAGRAMA DE SECUENCIA

Muestra los **mensajes** que se intercambian los **objetos** en tiempo de ejecución y para una tarea específica (escenario).

UML → **Diagramas de Interacción** (muy similares):
1) Secuencia (énfasis en el orden en que se envían los mensajes)
2) Colaboración (énfasis en la estructura, qué objetos se están relacionando con qué)

**Componentes**:

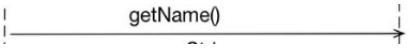
- **Objetos:**



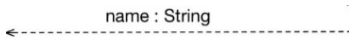

- **Mensajes:**

Modelan ***acciones*** del tipo:

1. **Call**: invocación de una operación de un objeto



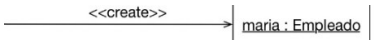

2. **Return**: valor de retorno de una operación




(Si es void No muestra nada )

3. **Create**: creación de un objeto

(Cuando se llama a New)

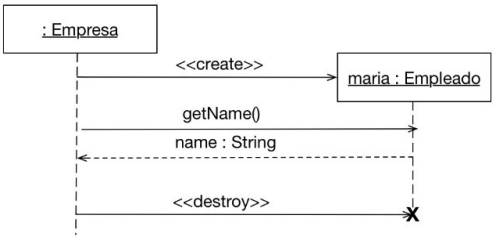



Todos los objetos que no sabemos cuando se crearon, se ponen en el mismo nivel (arriba de todo)

Los objetos que se van creando se ponen en el nivel del mensaje New Constructor

4. **Destroy**: destrucción de un objeto


5. <<**self**>> se llama a si mismo

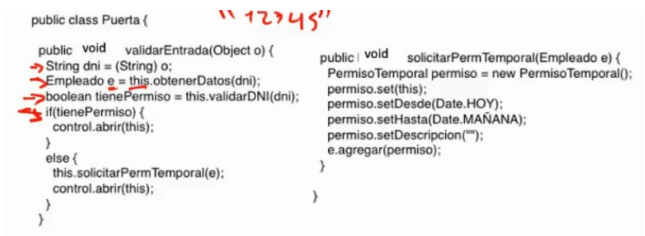



Todos los llamados (todo lo que tenga paréntesis), que aparecen adentro de una clase, son mensajes que se originan en esa clase, tienen que salir desde ese objeto.

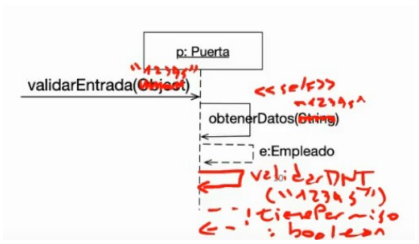
Para el receptor, me tengo que fijar de qué tipo es el objeto con el que se llama Ej: maría.getNombre(), maría es de tipo Empleado.

Pasos a seguir: 1) Identificar todos los mensajes (los que tienen () y los new)
2) Los métodos que no están detallados no me interesa mostrarlos.

Ejemplo:



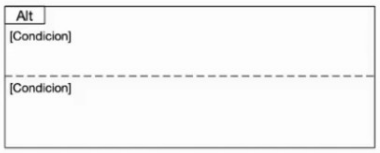
```
public class Puerta {
  public void validarEntrada(Object o) {
    String dni = (String) o;
    Empleado e = this.obtenerDatos(dni);
    boolean tienePermiso = this.validarDNI(dni);
    if(tienePermiso) {
      control.abrir(this);
    }
    else {
      this.solicitarPermTemporal(e);
      control.abrir(this);
    }
  }

  public void solicitarPermTemporal(Empleado e) {
    PermisoTemporal permiso = new PermisoTemporal();
    permiso.set(this);
    permiso.setDesde(Date.HOY);
    permiso.setHasta(Date.MAÑANA);
    permiso.setDescripcion("");
    e.agregar(permiso);
  }
}
```



**FRAMES** → Para representar if/else, while, for

**<u>Operadores</u>**:

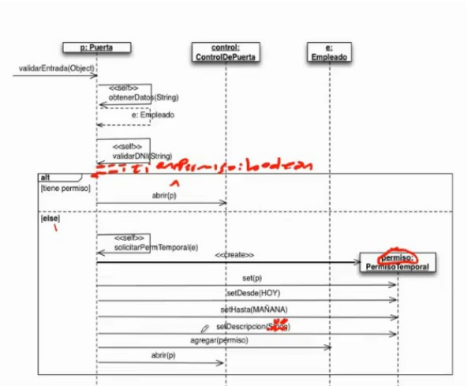
1. **Alt**: (para if-else, case) es para múltiples fragmentos. Solo ejecuta el de la condición verdadera



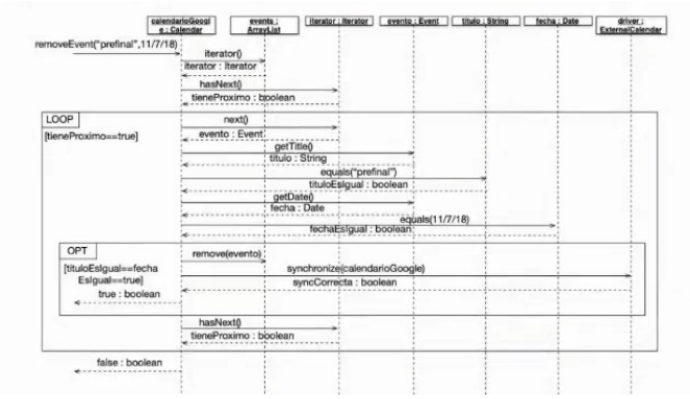

2. **Opt**: (para if) un único fragmento que solo ejecuta cuando la condición es verdadera


3. **Loop**: (para while, for, foreach) es un fragmento que se ejecuta múltiples veces

En el ejemplo:




**ojo → El retorno no es obligatorio que tenga el mismo nombre de variable, lo importante es el TIPO.**
**En el if → Primero se resuelven los llamados del if(cond..) y luego se hace el frame con el resultado de la condición.**




**Para el PARCIAL estar atento a: Diagrama CU y Diagrama de CLASES**

**1) Las epicas escribirlas como US**
**2) Las US CON Criterios de aceptación.**
**3) Refinar epicas → MINIMO 2 US**
**4) Casos de Uso → El Más importante. Si está mal se desaprueba directamente.**
**5) las inclusiones deben ser incluidas en al menos 2 CU, A MENOS que sea un CU en si mismo.**
**6) OJO con los CU sin peso (tienen dos líneas...ej imprime un ticket) y CU volando → SIN RETORNO**
**7) Link activación SOLO e/ actores y CU. No e/ CU y NO e/actores → SIN RETORNO**
**8) chequear que al final del CU se termine haciendo la funcionalidad → Ej: "Se almacena el pedido del cliente"**
**9) Agregación y Composición SOLO si tenemos CONTEXTO.**
**Analizar el orden de las preguntas: a) tiene atributo o colección de otra clase?**
**b) una clase representa el todo y la otra la parte?? (en enunciado "esta formado por…"**
**c) El tiempo de vida de la parte está ligado con el tiempo de vida del todo?**
***<u>Si respondo a) y c) → SI → es Asociación, porque no respondí Si a Agregación.</u>***

**10) Solo las ESTRUCTURALES llevan adorno.**
**11) this no es parte del mensaje, en parámetro se reemplaza por el nombre del objeto y si no tiene, por el tipo.**