# Disciplina Análisis y Diseño

El objetivo de esta disciplina es trasladar los requerimientos del sistema a una arquitectura del software con el fin de guiar la implementación.

Esta disciplina esta conformada por tres modelos:

* Modelo de Análisis
  + Diagrama de Clases de Análisis
  + Diagrama de Secuencias de Análisis
* Modelo de Navegación
  + Prototipo
  + Diagrama de Estado
  + Estimación del esfuerzo de desarrollo utilizando puntos de función
* Modelo de Diseño
  + Diagrama de Clases de Diseño
  + Diagrama de Secuencia de Diseño

## Modelo de Análisis

Refina los detalles del diagrama de clases del Negocio y de la disciplina de requerimientos. Se generan dos artefactos:

* Diagrama de Clases de Análisis
* Diagrama de Secuencia de Análisis

### Diagrama de Clases de Análisis

Este diagrama representa la estructura estática (Organizacional) del sistema, con las clases, atributos, operaciones y relaciones que van a ser diseñadas e implementadas luego. Aquí hablamos de operaciones (no métodos).

Aquí tenemos distintas categorías de clases, para diferenciarlas se usan los estereotipos

**<< algo >>**, a saber:

1. **Clases de Entidad**:

* Son las que vienen del diagrama de clases del negocio, representan los objetos de modelo.
* Su representación es igual a como se hacia en Negocio, solo que ahora se incluyen las operaciones si es necesario y los nombres de los atributos son mas en terminología de sistema.
* Para conseguir estas clases tomamos el Diagrama de Clases de Negocio y vemos cuales clases se trasladan tal cual y cuales pueden pasar a ser atributos
* Por ejemplo:

|  |  |
| --- | --- |
| **Negocio** | **Análisis** |
|  |  |

* Noten que año de creación ahora se llama “ac”, que hay métodos y que la imagen que antes era una clase, ahora es un atributo de obra

1. **Clases de Interfaz**:

* Se representan con el estéreo tipo **<< b >>**
* Su nombre debe comenzar con I, por ejemplo IObra
* Se identifican a partir de las interacciones entre los actores y los UC
* Se pueden ver como las futuras ventanas (pantallas) del sistema
* Sus atributos se sacan a partir del flujo básico y/o alternativo del UC
* Sus atributos son los elementos gráficos de la pantalla: botones, select, inputs, etc. Todos los que se pueden deducir de la descripción del UC
* No se suele colocarse métodos a nivel de análisis
* Ejemplo:

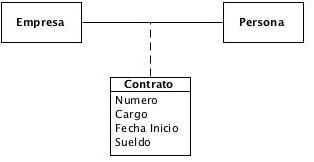
|  |  |
| --- | --- |
| **Clase Interfaz** | **Pantalla Prototipo** |
|  |  |

1. **Clases de Control**:

* Se representan con el estéreo tipo **<< ctrl >>**
* Su nombre debe comenzar con Ctrl, por ejemplo Ctrl\_Obras
* Se definen para evitar que las clases de Interfaz tengan relación con las de Entidad
* Cuando estas clases son implementadas, en ellas se definen las decisiones, el orden de instanciaciones, qué se hace cuando se instancia un elemento gráfico, etc.
* Sus métodos se sacan a partir de un pseudo-código

También tenemos nuevos elementos:

* **Clases asociativas**: una asociación entre dos clases, donde la propia asociación puede tener atributos y operaciones.



* **Asociaciones calificadas**: se usa un calificador para reducir la cardinalidad de una asociación N: N o 1: N



### Diagrama de Secuencias de Análisis

Los nuevos objetos del diagrama de clase se representan de la siguiente manera:

|  |  |
| --- | --- |
| Entidad: |  |
| Control: |  |
| Interfaz: |  |

Los nuevos mensajes son:

|  |  |
| --- | --- |
| De creación: |  |
| De destrucción: |  |

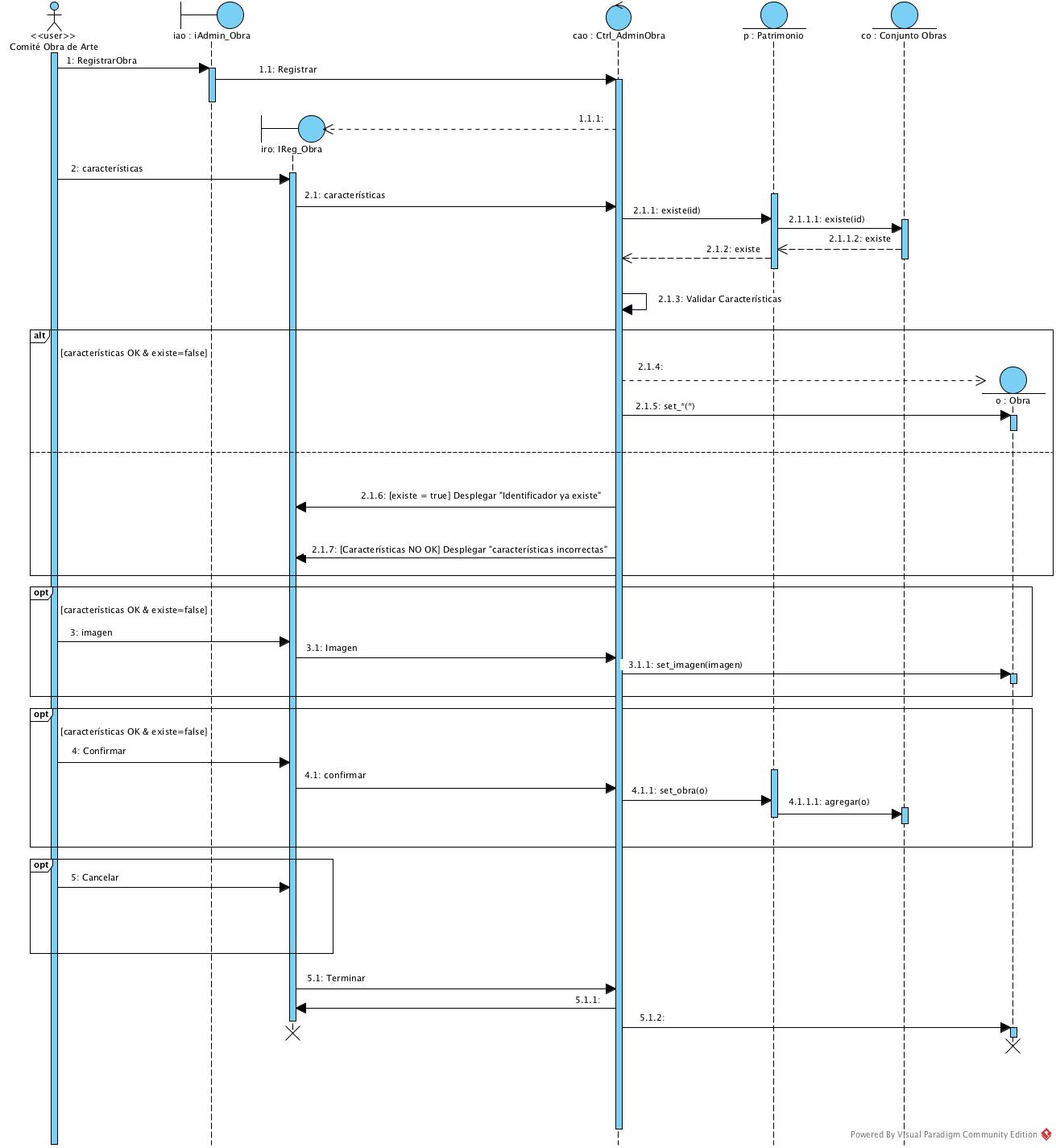
En los diagramas de Secuencia de Análisis representamos:

* Cómo se crean y destruyen las clases. Representamos cuando se instancia un objeto por primera vez, ya no todos surgen desde arriba.
* Vemos surgir las clases de control
* Si llamo un método de una clase en el diagrama de secuencia, ese método debe existir en el diagrama de clases

Aquí hay varias reglas:

* 1. El diagrama de secuencia debe representar lo que se describió en el UC
  2. Las relaciones entre clases son:
     + El actor se relaciona **solo** con clases interfaz
     + Las clases interfaz **solo** con las clases de control
     + Las clases interfaz **nunca** le mandan mensajes al Acto
     + Las clases de entidad **solo** con las de control
     + JAMAS hay clases de interfaz yendo directo con las de entidad
  3. Las clases de interfaz NO
  4. El actor no interactúa en la mitad de un fragmento combinado alt (lo que es equivalente a decir en la mitad de un if)

Ejemplo:



## Modelo de Navegación

Describe la secuencia de navegación que recorre un actor del sistema. Este modelo esta conformado por dos artefactos:

* Prototipo
* Diagrama de Estado

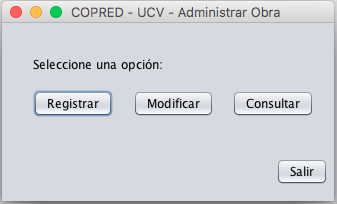
### Prototipo

* En el prototipo muestro elementos de la interfaz gráfica de las ventanas del sistema.
* Se hace en base a las clases de interfaz del diagrama de clases de Análisis.
* Cada pantalla se nombra igual que la clase de interfaz a la que representa
* Cuando creo la pantalla pueden surgir botones nuevos que no estaban en la clase, como por ejemplo “Salir”
* Para crear el prototipo se recomienda usar de una vez NetBean y crear las imágenes de las pantallas para insertar en el documento.
* Para crear las pantallas en netBean se usa: new —> JFrame Form

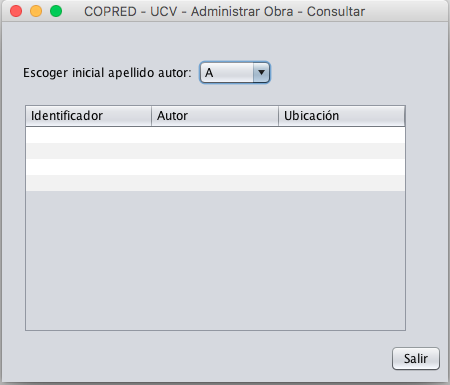
Varios detalles a tomar en cuenta en las pantallas

1. Agregar los botones finalizar, volver
2. Colocar títulos correctos, en plural si son consultas
3. Colocar etiquetas correctas (“Seleccione una opción:”)
4. Indicar donde van a ser desplegados los mensajes de error

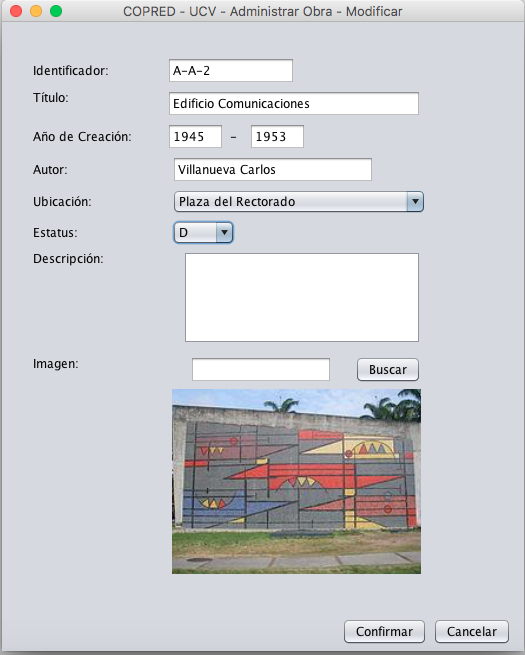
Ejemplo de pantallas:



**Figura 2. Prototipo I\_Admin\_O**



**Figura 3. Prototipo IObras**



**Figura 4. Prototipo IModO**

### Diagrama de Estado

El propósito de este diagrama es mostrar diferentes estados de un objeto con comportamiento dinámico. En nuestro caso lo usamos para ver las transiciones entre las ventanas del sistema, para representar la secuencia de navegación entre las ventanas.

Elementos de este diagrama:

|  |  |
| --- | --- |
| Estado Inicial: |  |
| Estado Final: |  |
| Estado intermedio: |  |
| Regla general de este diagrama |  |

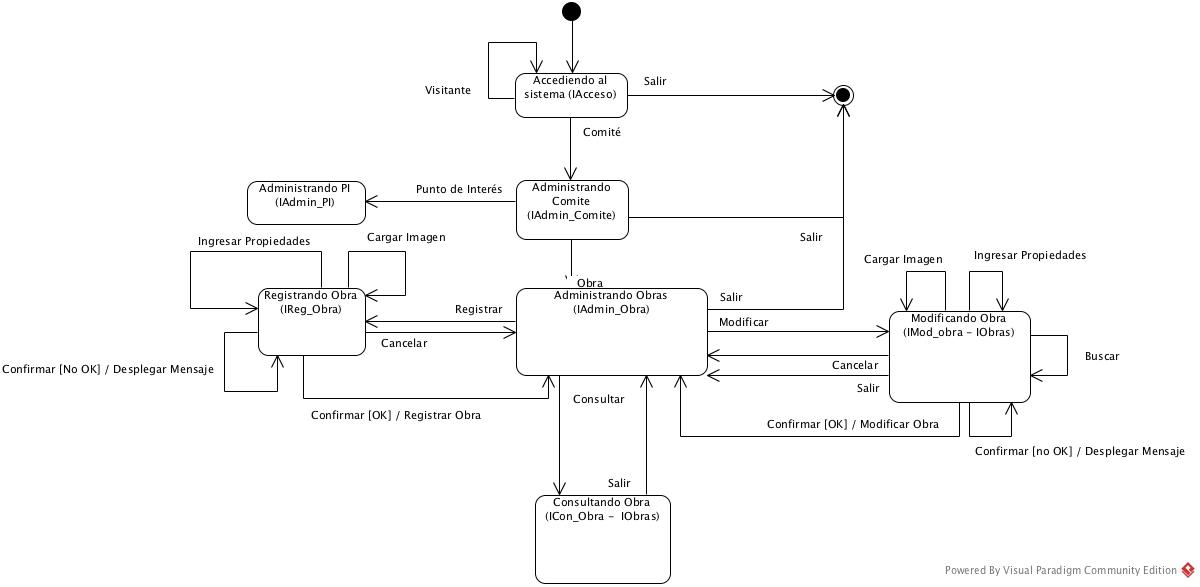
En este diagrama:

* Cada Estado es una **Ventana** y su nombre debe estar en gerundio:
* Cada Transición es un **posible camino** de navegación
* Los eventos son los **botones** y solo colocamos el nombre

Para hacer este diagrama nos basamos en el diagrama de clases de Análisis y tomamos en cuenta que:

1. Los nombres de los eventos deben ser los nombres de los botones del prototipo y con lo dicho en los diagrama de secuencia
2. “desplegando” no es un término para utilizar en el nombre de estado, es “Accediendo al sistema (Iacceso)”

Ejemplo:



### Estimación del esfuerzo de desarrollo utilizando puntos de función

* Tomamos todas las pantallas del prototipo y calculamos los puntos de función como hicimos en la práctica #2
  1. Identificar y cuanticar cada uno de los elementos de interés: entrada externa, salida externa, consulta externa, archivos interfaz externos y archivos lógicos internos.
  2. Para asignar el nivel de complejidad subjetiva de los elementos de interés, utilice las tablas 1, 2, 3 que se encuentran en el Anexo 2 de la práctica
  3. Determine el peso de los elementos de interés entrada externa, salida externa y consulta externa. En base al nivel de complejidad subjetiva que obtuvo en la parte (b), utilice los pesos que se indican en la Tabla 4 de la práctica #2 y para determinar el peso de los elementos de interés archivos interfaz externos y archivos lógicos internos utilice las tablas 5 y 6 de la práctica #2
  4. Calcule los puntos de función no ajustados
  5. Estime el esfuerzo de desarrollo de la aplicación para el registro de los equipos adquiridos y el investigador responsable de los equipos. El esfuerzo de desarrollo (ED) se expresa en persona-meses
  6. Calcule el factor de complejidad técnica (TCF) y de manera subjetiva evalúe en una escala de 0 a 5 a cada uno de los 14 subfactores técnicos que usted considere como irrelevante o no influye (0), incidental (1), moderado (2), promedio (3), significante (4) o esencial (5)
  7. Calcule el número de puntos de función (FP)
  8. Asuma que la aplicación se desarrolla en Java y convierta FP en líneas de código (LOC) para calcular el esfuerzo de desarrollo
* Para hacer esto tenga en cuenta que:
  + Hay interfaces reutilizables (muy parecidas: IObrMostrar, IEditObra), en esas solo se deberían contar una vez, exceptuando los botones que son diferentes, sobre todo si se considera la reutilización como uno de los factores técnicos
  + Asignar el nivel de complejidad subjetiva a los elementos de interés y peso:
    - Entrada externas: son entradas muy simples, por lo que la complejidad es baja.
    - Salidas externas: bajo
    - Consultas externas: promedio
    - EIF: bajo
    - ILF: bajo SON ARCHIVOS PLANOS MUY SIMPLES
  + Conversión del Nº de puntos de función en LOC utilizando java (La taza de conversión de es 14 LOC por punto de función

## Modelo de Diseño

En este diseño ya bajamos a nivel de código, en él:

* Especifica el diseño detallado de las clases, incluido estructuras y objetos de Java.
* Se incorpora conceptos OO.

En este modelo se generan dos artefactos:

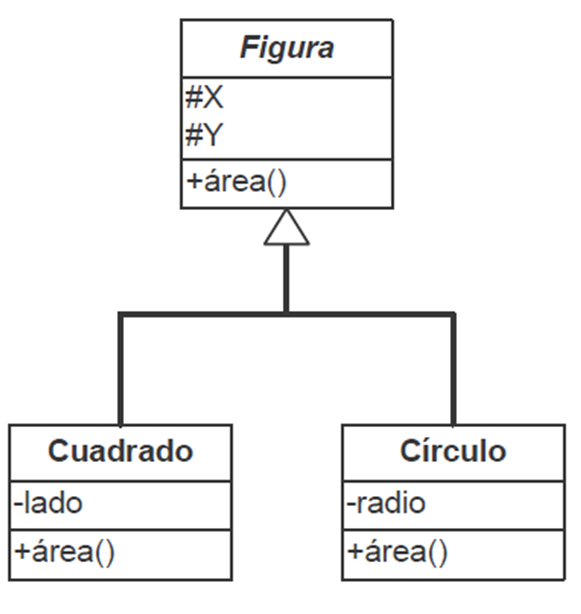
1. Diagrama de Clases de Diseño
2. Diagrama de Secuencia de Diseño

### Diagrama de Clases de Diseño

En este diagrama se agregan nuevos objetos:

1. **Clases de Abstractas**:

* El nombre va en itálica
* Son clases que no pueden tener instancias directas (no se puede hacer new sobre ellas)
* Deben tener al menos una subclase concreta que las instancie
* Puede tener métodos y atributos (protegidos) que son comunes en las subclases
* Puede tener métodos abstractos o concretos
* Ejemplo: la clase Figura del siguiente diagrama

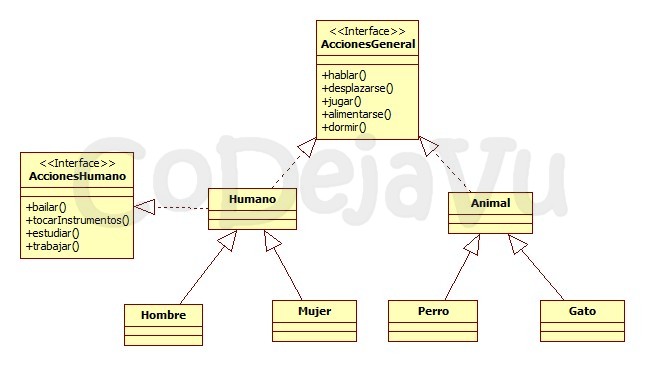


1. **Clases Concretas**:

* Pueden tener instancias directas
* Implementan los métodos abstractos de su clase abstracta padre
* No pueden tener métodos abstractos
* Ejemplo: las clases cuadrado y circulo del diagrama de arriba

1. **Objetos Interfaz**:

* Se identifica con el estereotipo **<< Interface >>**
* Solo tiene métodos abstractos
* Al especificar los métodos indicamos tipo de retorno, parámetros, solo se pone la signatura del cuerpo.
* No tiene atributos
* Sus subclases están obligadas a implementar todos los métodos abstractos que posee
* Debe existir al menos una clase que implemente la interfaz
* Se puede agregar notas con seudo-código cuando se las especifica en el diagrama
* NO confundir con las clases **<< boundary >> o << b >>**
* Ejemplo:



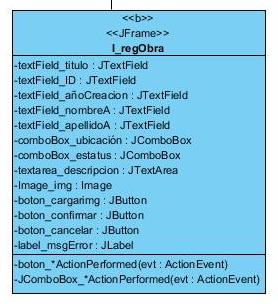
En este diagrama se incorpora la visibilidad y el tipo de los atributos y métodos

+ publico : cualquier otra clase puede verlo y modificarlo

- privado : solo es accedido por la misma clase, la subclase no lo hereda

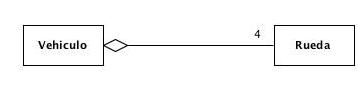
# protegido : solo es accedido por él y sus subclases, se hereda

Ejemplo de uso:



En este diagrama se incluye un nueva relación, la de agregación:

* Esta relación es del tipo compuesto / componente
* Es transitiva: si A —> B —> C, entonces A —> C
* Es antisimétrica: si A —> B , B —X—> A (NO)
* Los componentes tienen existencia propia
* La relación tiene sentido de dirección
* Ejemplo:



Donde:

Vehículo = compuesto

Rueda = componente

La relación = Vehículo “compuesto por” 4 Ruedas

El sentido de dirección de la relación en Vehículo —> Rueda

Para realizar este diagrama tomamos el diagrama de clases de análisis y:

1. Incorporamos las cosas que aparecieron en el prototipo y no estaban en el diagrama de clases de diseño, por ejemplo algunos botones
2. Revisamos las operaciones de análisis y vemos cuales se transformar en métodos, si son los mismos, etc
3. Hay que agregar los métodos privados asociados a los eventos de cada atributo en las clases, los get y set
4. Revisar que clases pueden ser abstractas y las definimos colocando en itálica el nombre
5. Revisamos si hay objetos Interfaz y las agregamos
6. Incorporamos estructuras de java, por ejemplo, las clases de interfaz pueden ser JFrame, los conjuntos listas
7. Creamos estructuras de datos para relaciones 1 a n (Jlistas)
8. Revisamos si es posible incluir algún patrón de diseño en algua clase, por ejemplo <<singleton>>, de ser así la identificamos con el estereotipo y agregamos los métodos y atributos propios del patrón
9. Incluimos relaciones de agregación de existir
10. Se especifican estructuras de datos para implementar relaciones 1:N

Otras revisiones:

1. De acuerdo al estándar, los nombres de atributos y métodos comienzan con minúscula, por ejemplo: titulo, get\_Atributos
2. Además, en clases de diseño, el nombre de una clase no puede tener espacios en blanco porque es el nombre que van a utilizar en la implementación
3. La visibilidad de los atributos en las clases interfaz no puede ser pública porque va en contra del principio de diseño OO de ocultamiento de información y encapsulamiento

### Diagrama de Secuencia de Diseño

En estos diagramas de secuencia se:

* Incluyen los nuevos objetos de Diseño, por ejemplo
  + Estructuras de datos de Java (ArrayList, JFrame) y se representan como en Java
* Al representar los intercambios de mensajes se baja de nivel, se hace ya con lenguaje y funcionamiento de Java
* Se muestra el intercambio de mensajes entre los objetos de diseño, solo si, agrega algo que no este en el diagrama de clases de análisis, por eso hay que analizar si es significativo realizar todos estos diagramas de secuencia porque son casi tal cual a los diagramas de análisis