1. Organización de la práctica

Este documento contiene la información necesaria para realizar la segunda práctica de la asignatura. Esta práctica tiene planificada una duración de dos sesiones, en las cuales se realizará el diseño del sistema software. A continuación se detallan los objetivos, organización y evaluación de la práctica:

Sesión 1: diagrama de clases

- Objetivos:
  1. Aplicar la técnica de diagramas de clase de UML para el diseño estructural del sistema.
  2. Avanzar en el conocimiento y desarrollo del sistema validando y refinando la especificación de requisitos.
- Preparación: no requiere preparación previa por parte del estudiante.
- Seguimiento y evaluación: al comienzo de la sesión se repasarán los conceptos de UML necesarios para realizar la práctica utilizando la herramienta *Visual Paradigm*. Durante el resto de la sesión, los estudiantes trabajarán en la elaboración del diagrama de clases. No hay evaluación específica durante esta sesión.

Sesión 2: diagramas de secuencia

- Objetivos:
  1. Aplicar la técnica de diagramas de secuencia de UML para el diseño del comportamiento del sistema.
  2. Avanzar en el conocimiento y desarrollo del sistema para poder abordar la fase de implementación y pruebas.

- Preparación: no requiere preparación previa por parte del estudiante.
- Seguimiento y evaluación: se trabajará en el aula para completar el diseño del sistema, elaborando como mínimo 3-4[1] diagramas de secuencia, así como resolver dudas. Tras esta sesión, los estudiantes dispondrán de unos días adicionales para realizar la entrega de la práctica en Moodle. Dicha entrega consistirá en incorporar al documento de prácticas los apartados correspondientes al diseño del sistema, incluyendo los diagramas de clases y secuencia. Además, se evaluará la planificación y el progreso en la plataforma *YouTrack / Trello*.

Esta práctica está directamente relacionada con el contenido teórico de la asignatura, en concreto con el Tema 5: técnicas de especificación y modelado. Por tanto, es deber del estudiante consultar y repasar dicho material durante la elaboración de la práctica. En el resto del documento se resumen solo aquellos aspectos esenciales para el desarrollo de la práctica y el uso de *Visual Paradigm*.

[1] Se deben elaborar, al menos, tantos diagramas como miembros del equipo.

2. Diagrama de clases

El diagrama de clases de UML [1] nos proporciona una notación gráfica para representar estructuras de información estática [2, 3]. Este tipo de diagrama es muy útil para diseñar sistemas orientados a objetos, pues nos permiten visualizar las clases, incluyendo sus atributos y operaciones, así como las relaciones entre ellas.

2.1. Elementos de un diagrama de clases

Una clase en UML se compone de la información que se indica a continuación. Posteriormente, la Figura 1 muestra como se visualiza esta información en *Visual Paradigm*.

- Nombre de la clase
- Indicación de si la clase es abstracta
- Atributos de la clase
  - Visibilidad: pública (+), privada (-), o protegida (#)
  - Nombre
  - Tipo de dato
- Operaciones de la clase
  - Visibilidad: pública (+), privada (-), o protegida (#)
  - Parámetros, detallando sus tipo de dato, nombre, y dirección (*in*, *out*, *inout*)
  - Tipo de dato de retorno

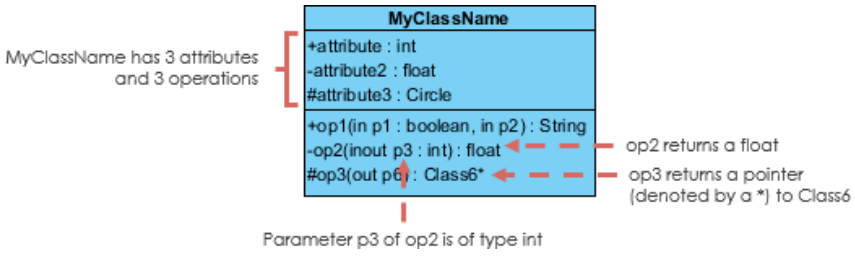
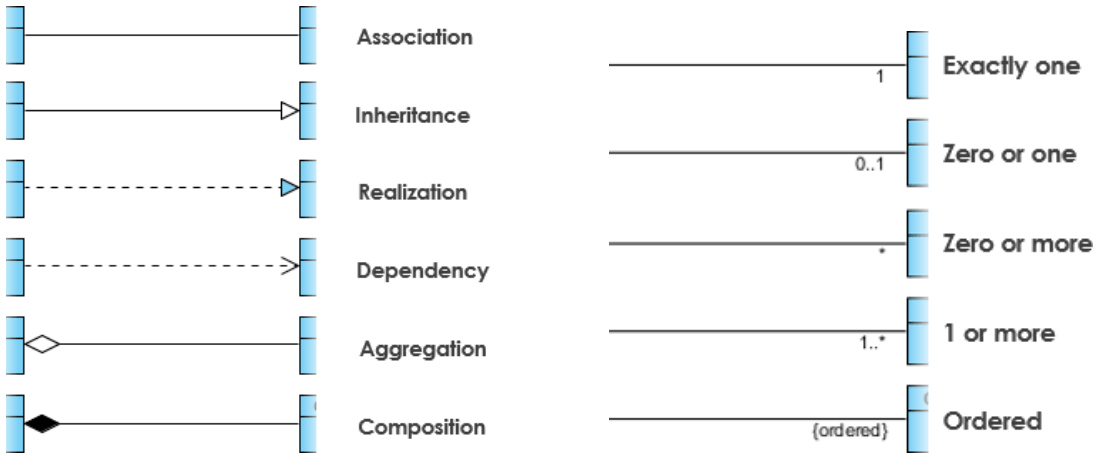


Figura 1: Representación de una clase en *Visual Paradigm*. Fuente: [3].

Por otro lado, las clases se relacionan entre sí por medio de alguna de las relaciones que se indican a continuación. La Figura 2 se muestra la representación gráfica de las relaciones entre clases y su cardinalidad (es decir, número de instancias de cada clase que participan en la relación) en *Visual Paradigm*.

- **Asociación**. Indica una conexión funcional entre dos clases. Es habitual indicar la cardinalidad con la que dicha relación sucede.
- **Herencia**. Representa una relación "*is-a*" entre una clase "padre" y una o varias clases "hijas".
- **Realización**. Es una relación entre una clase que indica cómo se debe realizar una implementación (estereotipada como interfaz) y las clases que implementan dicha funcionalidad.
- **Dependencia**. Es un tipo de asociación que establece que la definición de una clase se ve influenciada por cambios en otra. Es una relación unidireccional.
- **Agregación**. Es un tipo especial de asociación que indica que una clase es parte de otra, pero tienen diferentes líneas de vida.
- **Composición**. Es un tipo especial de agregación donde la clase que forma parte de la otra no puede existir por sí misma.



(a) Relaciones (b) Cardinalidad

Figura 2: Representación de los distintos tipos de relaciones entre clases y su cardinalidad en *Visual Paradigm*. Fuente: [3].

2.2. Diagrama de clases en *Visual Paradigm*

Para crear un diagrama de clases en *Visual Paradigm* seguiremos los siguientes pasos:

1. Crear un nuevo proyecto, o abrir uno creado previamente.
2. En la zona de trabajo, seleccionar *System Modeling*. Ir a la columna UML y pulsar el botón '+'.
3. En el cuadro de diálogo, seleccionar el tipo de diagrama *Class Diagram*. A continuación, seleccionar la opción *Blank* para crear un diagrama vacío. Introducir un nombre para el diagrama.
4. Tras estos pasos, se nos mostrará el editor de diagramas de clases. En el panel lateral tenemos los diferentes elementos que pueden aparecer en este tipo de diagrama (clases, relaciones, etc.).

Una vez arrastrados los elementos del diagrama, debemos completar la especificación de cada clase accediendo con el botón derecho a la opción *Open specification* o pulsando sobre el icono con forma de lupa en la esquina superior. Se abrirá un editor para indicar las características de la clase, y varias pestañas para definir sus atributos y operaciones. Tras guardar los cambios, la información aparecerá en la clase. Si el diagrama contiene muchas clases, podemos ocultar las franjas de atributos y operadores pulsando el icono '-' que aparece en cada franja.

Para crear las relaciones, debemos seleccionar el tipo concreto y trazar una línea entre las clases que deben relacionarse. A continuación, deberemos indicar la cardinalidad, descripción de la relación, y la navegabilidad bidireccional (*true/false*), si es necesario. Para ello haremos doble *click* sobre la línea. Aparecerá un cuadro de texto para escribir y dos símbolos de flecha para editar los extremos de la relación. También ahí se podrá especificar si la asociación es una agregación (*shared*) o una composición. La Figura 3 muestra un diagrama de clases completo en *Visual Paradigm*.

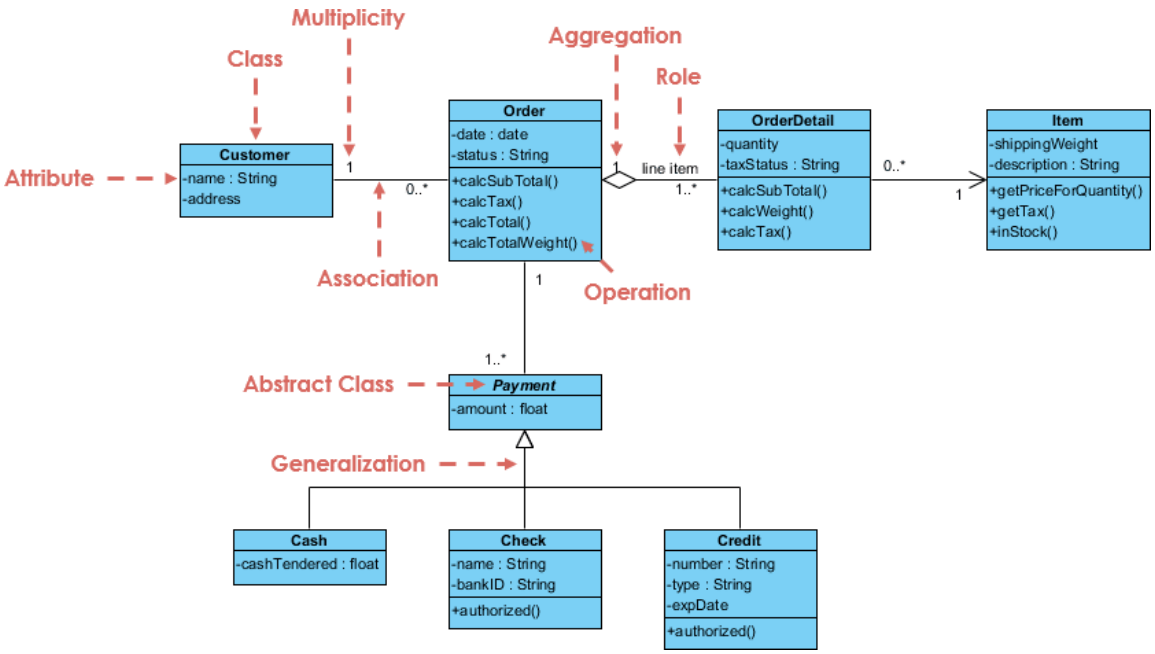


Figura 3: Diagrama de clases en *Visual Paradigm*. Fuente: [3].

2.3. Plantilla para la especificación de la clase

Para la especificación de las clases en la documentación se utilizará la plantilla mostrada en la Tabla 1. De esta forma, la información será más legible y completa que si se muestra en el diagrama de clases. Por tanto, puede utilizarse la forma “colapsada” del diagrama en la documentación, o incluir tan solo la información básica.

Tabla 1: Plantilla para especificar una clase.

Clase: Nombre de la clase			
Descripción de la clase			
		Atributos	
+/-/#	Nombre atributo 1	Tipo atributo 1	Descripción atributo 1
+/-/#	Nombre atributo 2	Tipo atributo 2	Descripción atributo 2
...		...	
+/-/#	Nombre atributo m	Tipo atributo m	Descripción atributo m
		Operaciones	
+/-/#	Nombre operación 1	Tipo retorno operación 1	Descripción operación 1
+/-/#	Nombre parámetro 1	Tipo dato parámetro 1	Descripción parámetro 1
+/-/#	Nombre parámetro 2	Tipo dato parámetro 2	Descripción parámetro 2
...		...	
+/-/#	Nombre parámetro n	Tipo dato parámetro n	Descripción parámetro n
+/-/#	Nombre operación 2	Tipo retorno operación 2	Descripción operación 2
...		...	
+/-/#	Nombre operación p	Tipo retorno operación p	Descripción operación p

3. Matrices de trazabilidad

Construir una matriz de trazabilidad es una técnica de validación que nos permite comprobar que los elementos diseñados dan respuesta a las necesidades expresadas en los requisitos. Por tanto, también ayuda a confirmar que todos los requisitos están cubiertos en el diseño.

En esta práctica se realizarán dos matrices de trazabilidad:

- Matriz de trazabilidad de requisitos funcionales frente a casos de uso. Cada requisito debe estar cubierto por, al menos, un caso de uso. Todo caso de uso debe dar respuesta a uno o más requisitos. Se recomienda realizarla al comienzo de la práctica, par así refinar los casos de uso si fuese necesario. Puede observarse un ejemplo en la Tabla 2.
- Matriz de trazabilidad de casos de uso frente a clases. Cada clase debe tener correspondencia con uno o varios casos de uso. Todo caso de uso debe tener al menos una clase asociada. Conviene realizarla tras terminar el diagrama de clases para comprobar que las clases son adecuadas.

Tabla 2: Ejemplo de matriz de trazabilidad (requisitos funcionales vs casos de uso).

	CU1	CU2	CU3	CU4	CU5
RF1	✓		✓		
RF2		✓			
RF3				✓	
RF4		✓		✓	
RF5	✓				
RF6					✓
RF7					✓
RF8		✓			

4. Diagrama de secuencia

El diagrama de secuencia de UML [1] es un tipo de diagrama de interacción que nos permite modelar el comportamiento del sistema [4]. En concreto, este tipo de diagrama permite especificar la colaboración entre objetos, o entre el usuario y el sistema, para realizar una operación a un nivel de abstracción elevado.

4.1. Elementos de un diagrama de secuencia

En un diagrama de secuencia se utilizan dos tipos de ejes. El eje horizontal es donde se posicionan los objetos que intervienen en la colaboración. El eje vertical, que representa el tiempo, es donde se ubican los mensajes intercambiados entre los objetos, siguiendo un orden cronológico. El resto de elementos que intervienen en un diagrama de secuencia son:

- **Actor**. Agente externo (usuario u otro sistema) que participa en la interacción.
- **Línea de vida**. Nace de un objeto del sistema que interviene en la colaboración, y es de donde parten o llegan los mensajes intercambiados entre los actores u otros objetos. El periodo durante el cual el objeto está activo se denomina “activación”, y se representa con un rectángulo delgado sobre la línea.
- **Mensaje**. Indica una comunicación entre dos líneas de vida. Puede representar la invocación de una operación, una llamada recursiva, la instanciación o destrucción de un objeto, o un mensaje de retorno. Deben ir numerados según su orden de ejecución.
- **Bloques**. Fragmento que encuadra una secuencia de mensajes con un significado o estructura especial. Permiten construir secuencias más complejas mediante bucles, flujos alternativos, etc. Los más habituales son:
  - *loop*. Un fragmento que se ejecuta varias veces.
  - *alt*. Un fragmento de secuencias alternativas, donde solo aquella que cumple la condición será ejecutada.
  - *opt*. Un fragmento opcional que solo se ejecuta si se cumple la condición. Equivalente a un bloque *alt* con un único carril.
  - *par*. Cada calle del fragmento se ejecutará en paralelo.

4.2. Diagrama de secuencia en *Visual Paradigm*

Para crear un diagrama de secuencia en *Visual Paradigm* seguimos los siguientes pasos:

1. Crear un nuevo proyecto o abrir uno creado previamente.
2. En la zona inferior, seleccionar *System Modeling*. Ir a la columna *UML* y pulsar el botón '+'.
3. En el cuadro de diálogo, seleccionar el tipo de diagrama *Sequence Diagram*. A continuación, seleccionar la opción *Blank* para crear un diagrama vacío. Introducir un nombre para el diagrama.
4. Tras estos pasos, se nos mostrará el editor de diagramas de secuencia. En el panel lateral tenemos los diferentes elementos que pueden aparecer en este tipo de diagrama (líneas de vida, mensajes, bloques, etc.).

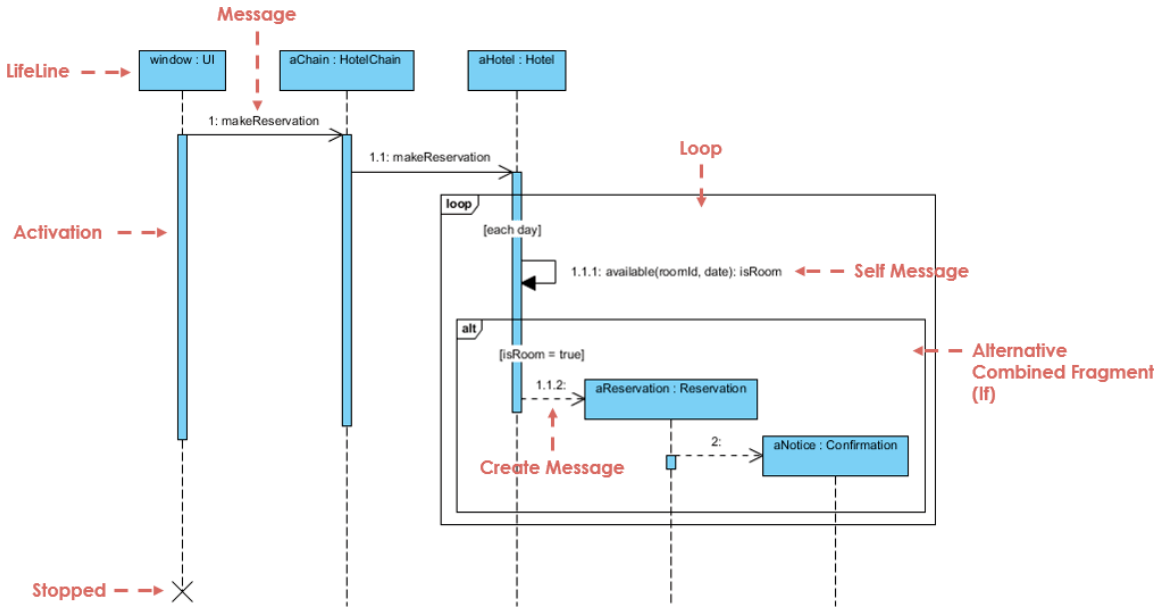


Figura 4: Diagrama de secuencia en *Visual Paradigm*. Fuente: [4].

Al igual que con el resto de diagramas, debemos arrastrar el elemento deseado al editor, donde podremos editar su nombre y propiedades. Para acceder a los distintos tipos de mensajes, hay que pulsar el triángulo negro que aparece en la parte inferior de este elemento. De forma similar, los bloques *loop* y *alt* se encuentran definidos en el mismo apartado. Las condiciones del bloque *alt* deben especificarse en *Open specification* →

Interaction Operands → *Guard*. Finalmente, las líneas de vida se extienden conforme se crean los mensajes entre objetos, mientras que los mensajes se numeran automáticamente según el orden temporal. La Figura 4 muestra un diagrama de secuencia creado con *Visual Paradigm*.

Referencias

[1] Jim Arlow and Ila Neustadt. *UML 2*. Anaya, 2006.

[2] Visual Paradigm. What is class diagram? Disponible en: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/.

[3] Visual Paradigm. UML class diagram tutorial. Disponible en: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.

[4] Visual Paradigm. What is sequence diagram? Disponible en: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/.