Elaboración de diagramas de clases

- Diapositivas
- Actividades

Introducción

En esta Unidad aprenderemos a:

- Identificar las herramientas para la elaboración de diagramas de clases.
- Interpretar el significado de diagramas de clases.
- Generar código a partir de un diagrama de clases.
- Generar un diagrama de clases mediante ingeniería inversa.

UML

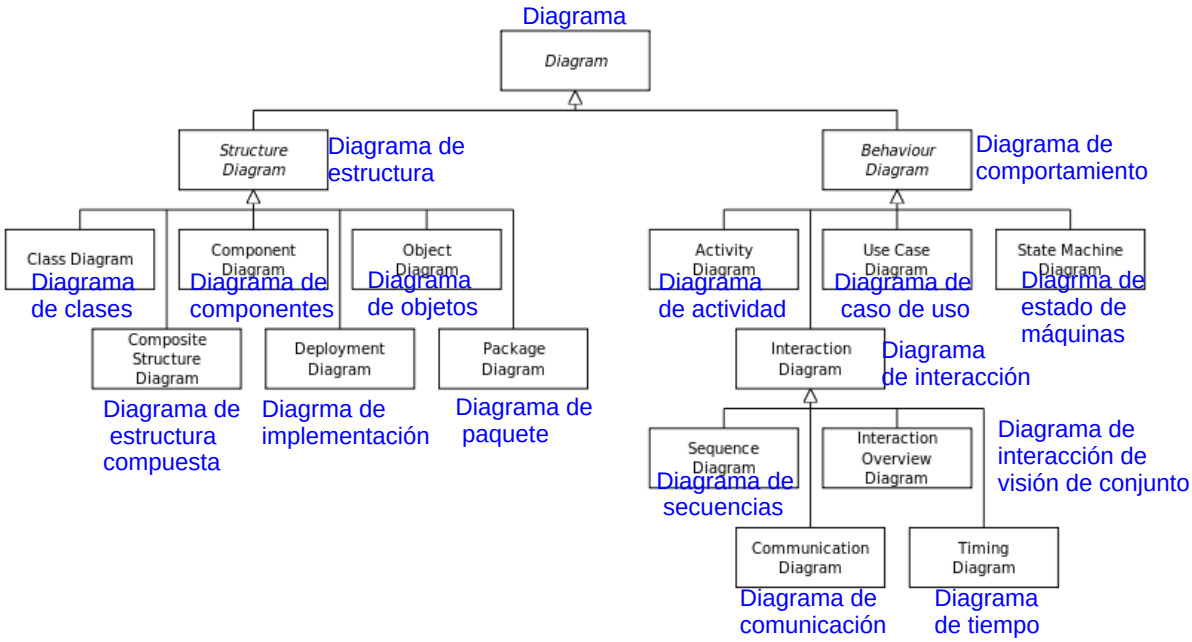
Lenguaje de modelado unificado

- Es un lenguaje visual de propósito general para representar **modelos**.
- Pretende proporcionar una forma estándar de representar el diseño de un sistema.
- Dispone de numerosos tipos de diagramas.
- Cada tipo de diagrama muestra un aspecto diferente del modelo.
- Actualmente disponible la versión 2.5. Existen algunas diferencias respecto a las versiones 1.x.

UML: Tipos de diagramas (I)

- **diagramas de estructura** (aspecto estático)
- **diagramas de comportamiento** (aspecto dinámico)

UML: Tipos de diagramas (II)

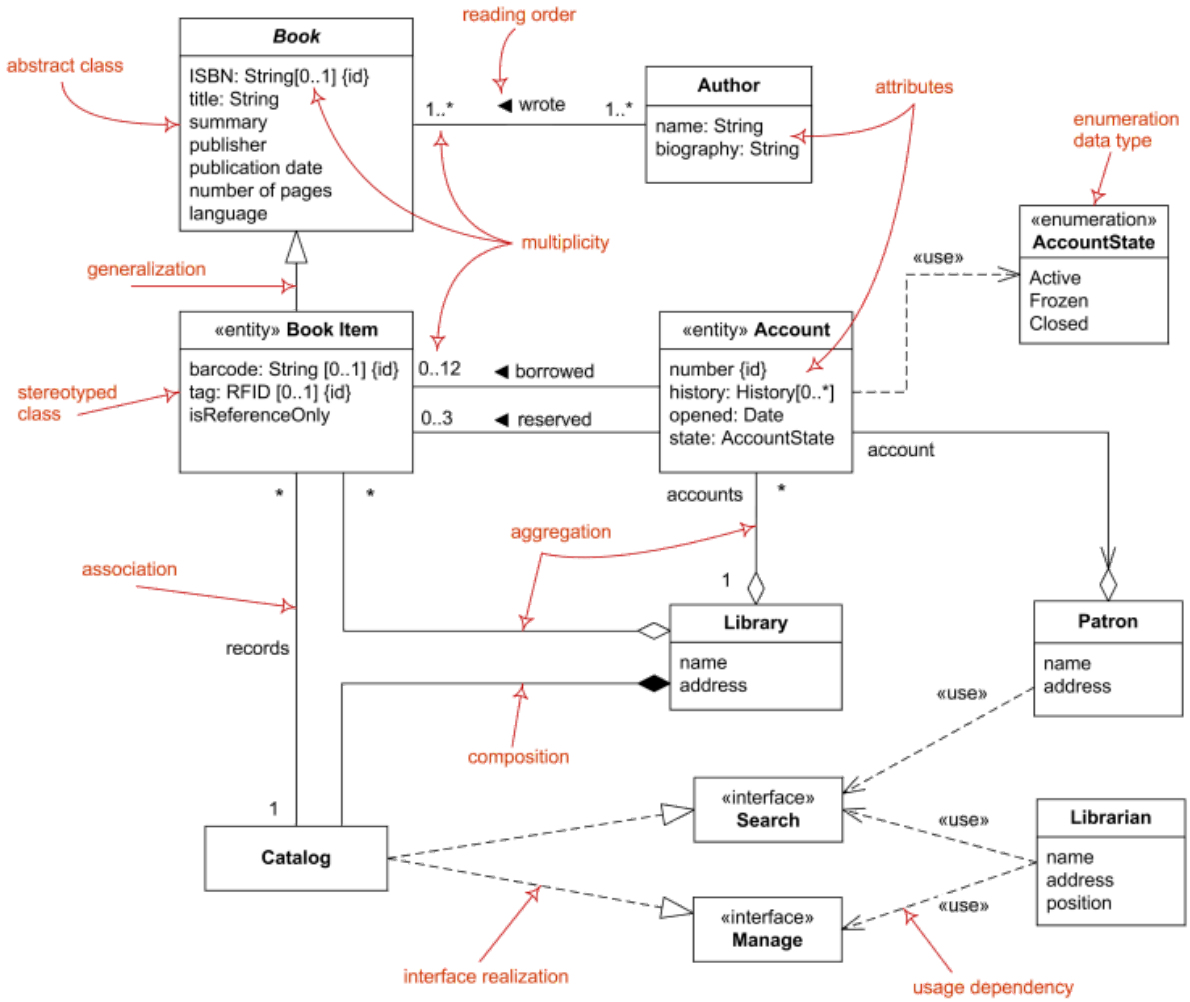


UML: Diagramas de estructura

Los más utilizados son:

- Diagramas de clases
- Diagramas de paquetes
- Diagramas de componentes
- Diagramas de implementación

Diagramas de clases

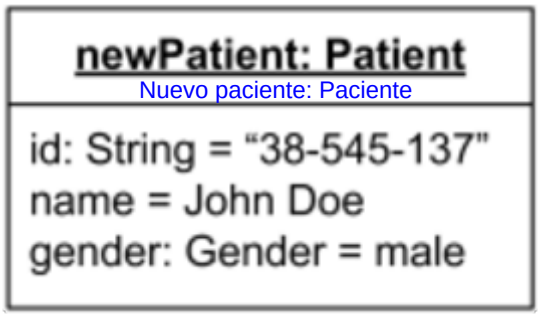


Clases

ELEMENTOS
- Clases
- Relaciones
  * Nombre
  * Multiplicidad
  * Roles

Objetos

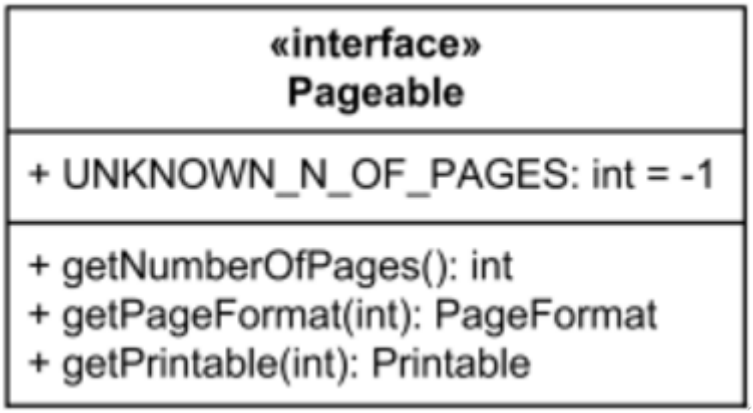
- No aparecen en los diagramas de clases.
- Aparecen en los diagramas de objetos y en diagramas de comportamiento.



Interfaces

<<estereotipo>>: los estereotipos permiten tomar elementos propios del UML y convertirlos en otros que se ajusten a las necesidades. Se usan cuando no existe tal elemento en UML.

- Un componente tiene su comportamiento definido en términos de las interfaces proporcionadas y las interfaces requeridas (potencialmente expuestas a través de puertos).



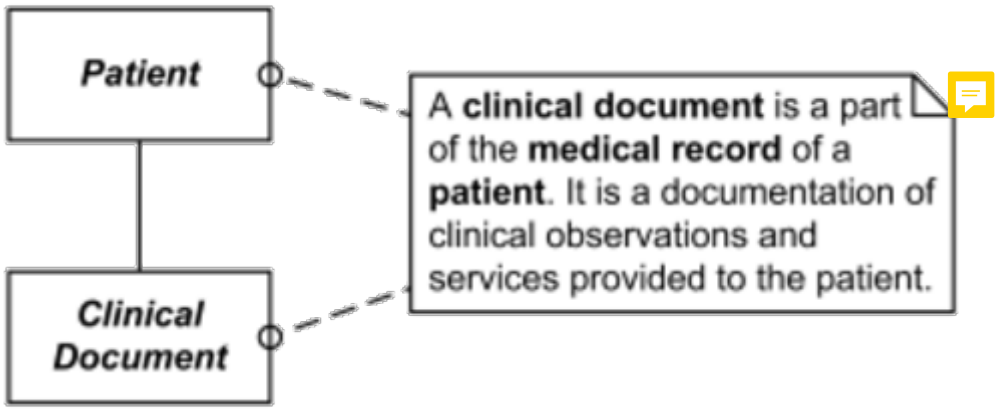
Relaciones

- Asociación
  - Agregación
  - Composición
- Dependencia
- Generalización / Especialización

- Realización

Asociación

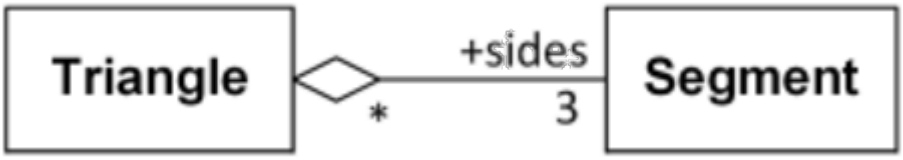
- Es el vínculo (conector) entre 2 clases que necesitan comunicarse entre sí.
- Se puede representar mediante una línea con una flecha que indica la dirección de navegación.
- En el caso de que la flecha esté en ambos lados, la asociación tiene asociación bidireccional.



Agregación

- La clase "padre" es contenedor no exclusivo de la clase "hija".
- Si eliminamos la instancia de la clase "padre" NO se elimina la instancia de la clase "hija".

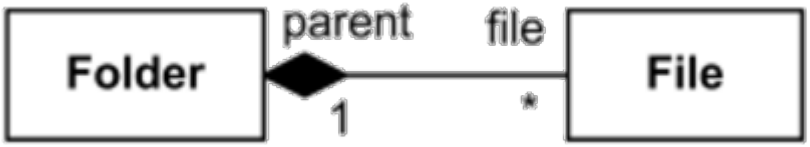
(Aquí los términos "padre" e "hija" no tienen ninguna relación con la herencia.



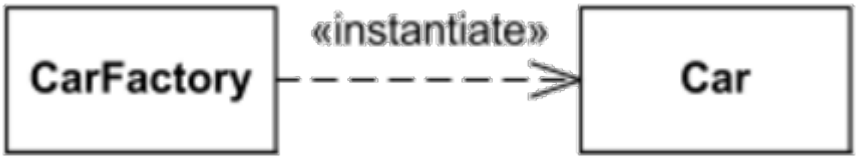
Composición

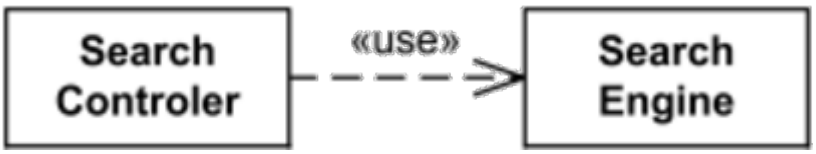
- La clase "padre" es contenedor exclusivo de la clase "hija".
- Si eliminamos la instancia de la clase "padre" SÍ se elimina la instancia de la clase "hija".

(Aquí los términos "padre" e "hija" no tienen ninguna relación con la herencia.)

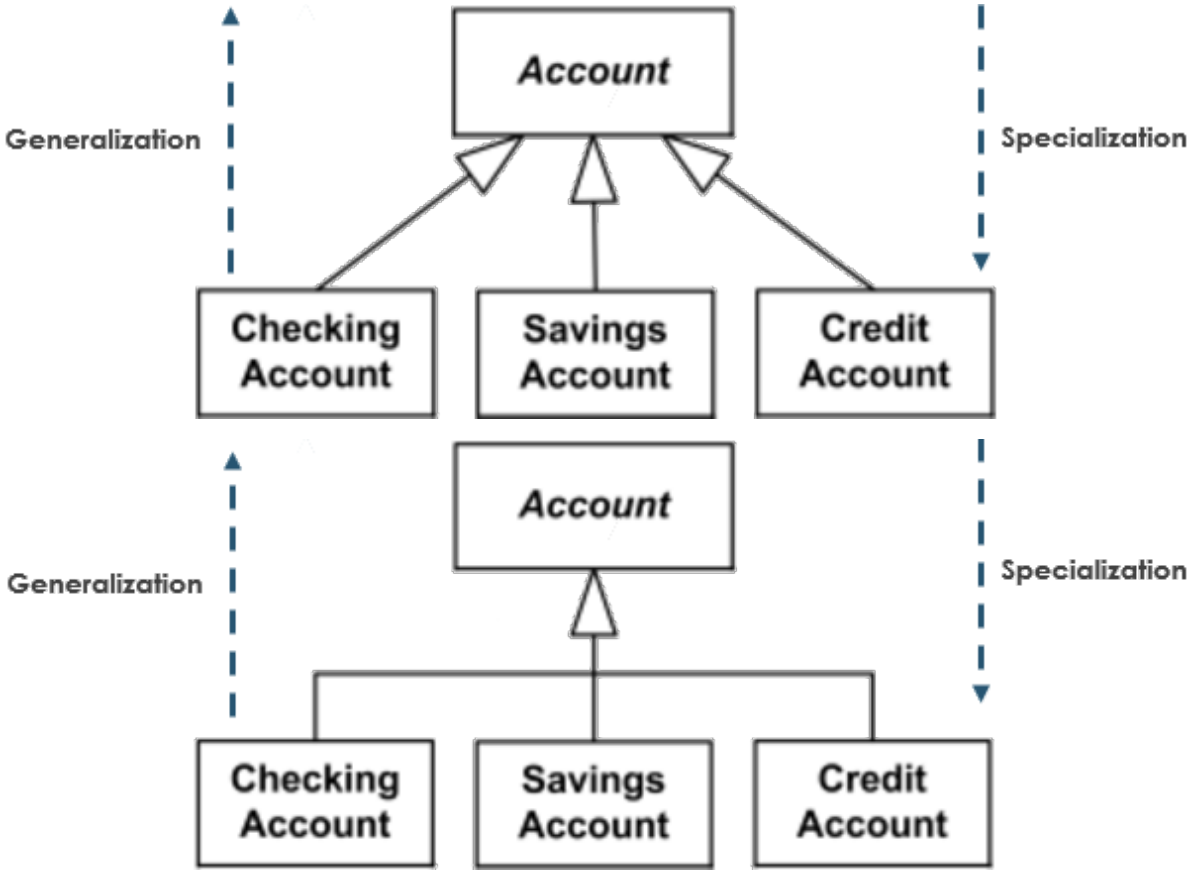


Dependencia

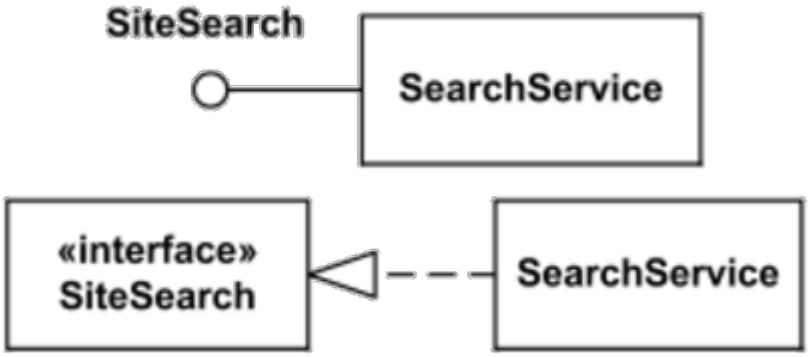




Generalización (herencia)

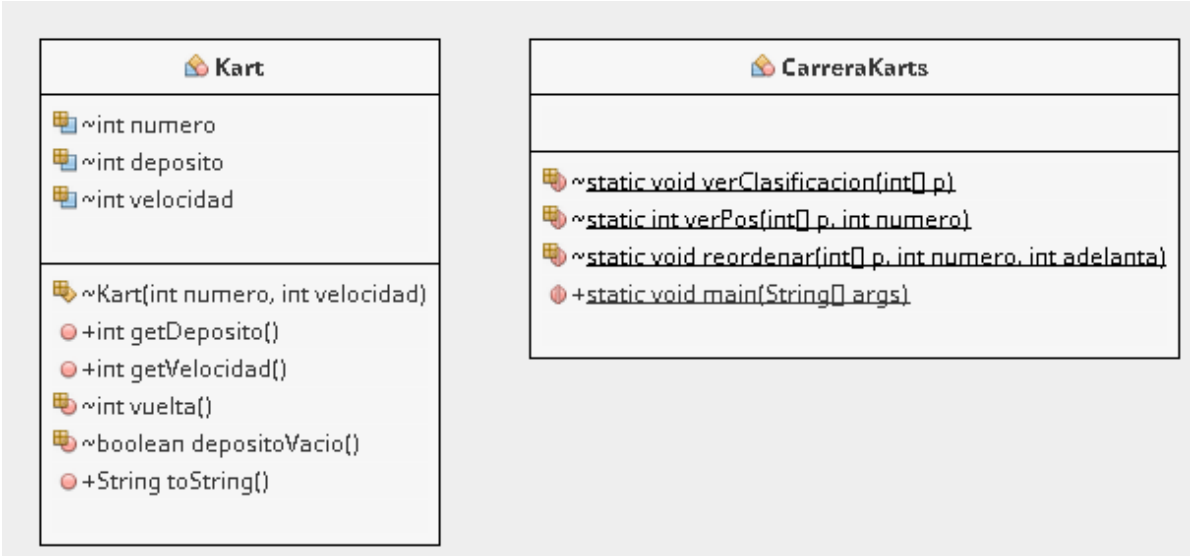


Realización (implementación de interfaces)

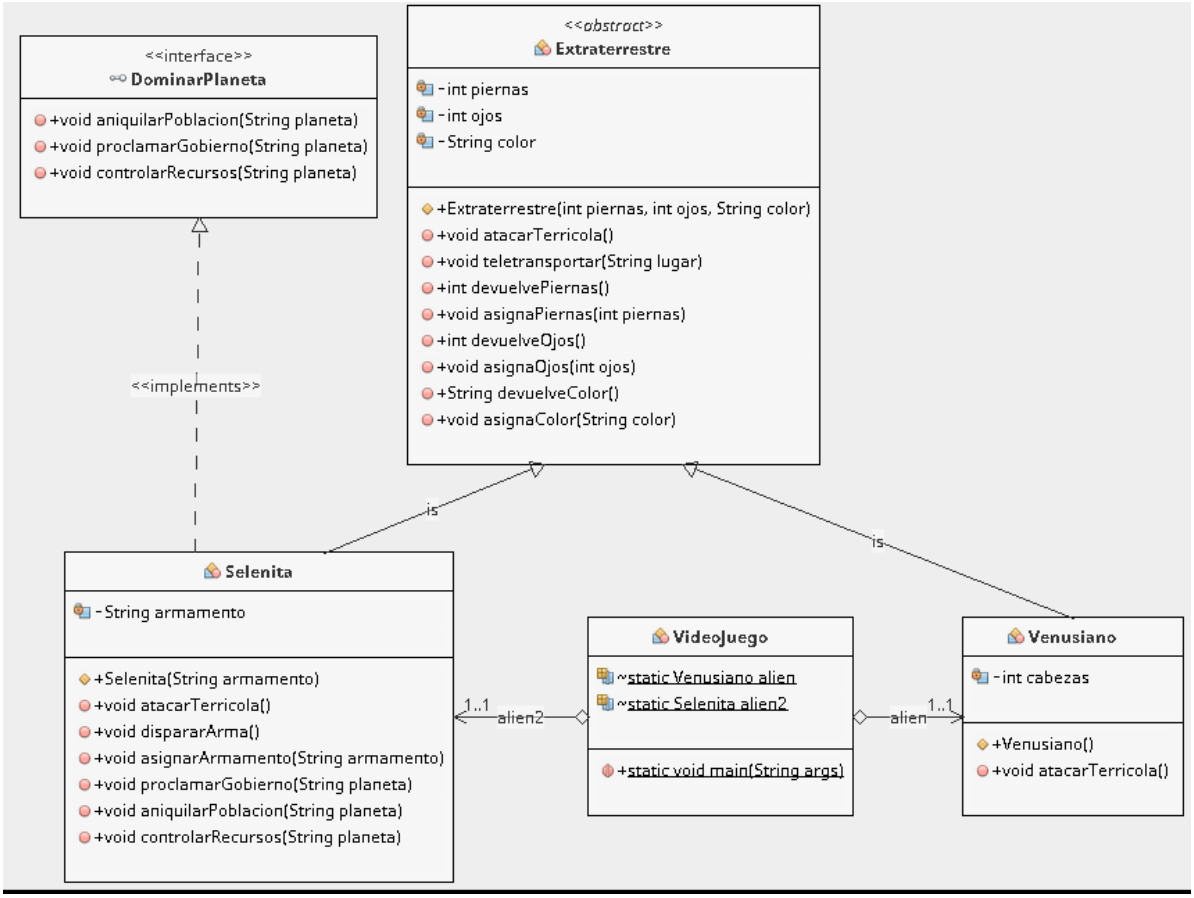


Ejemplos

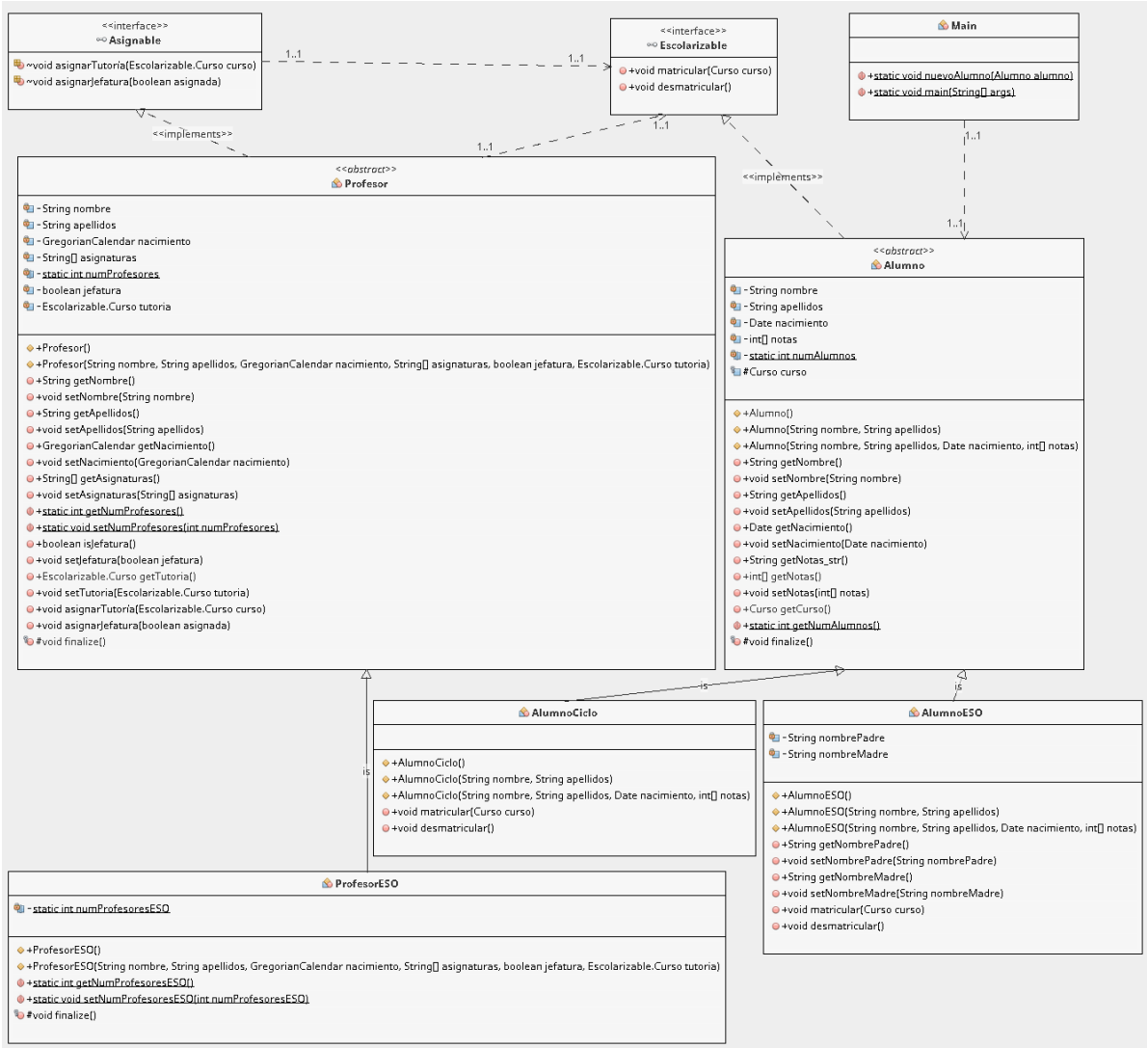
Karts



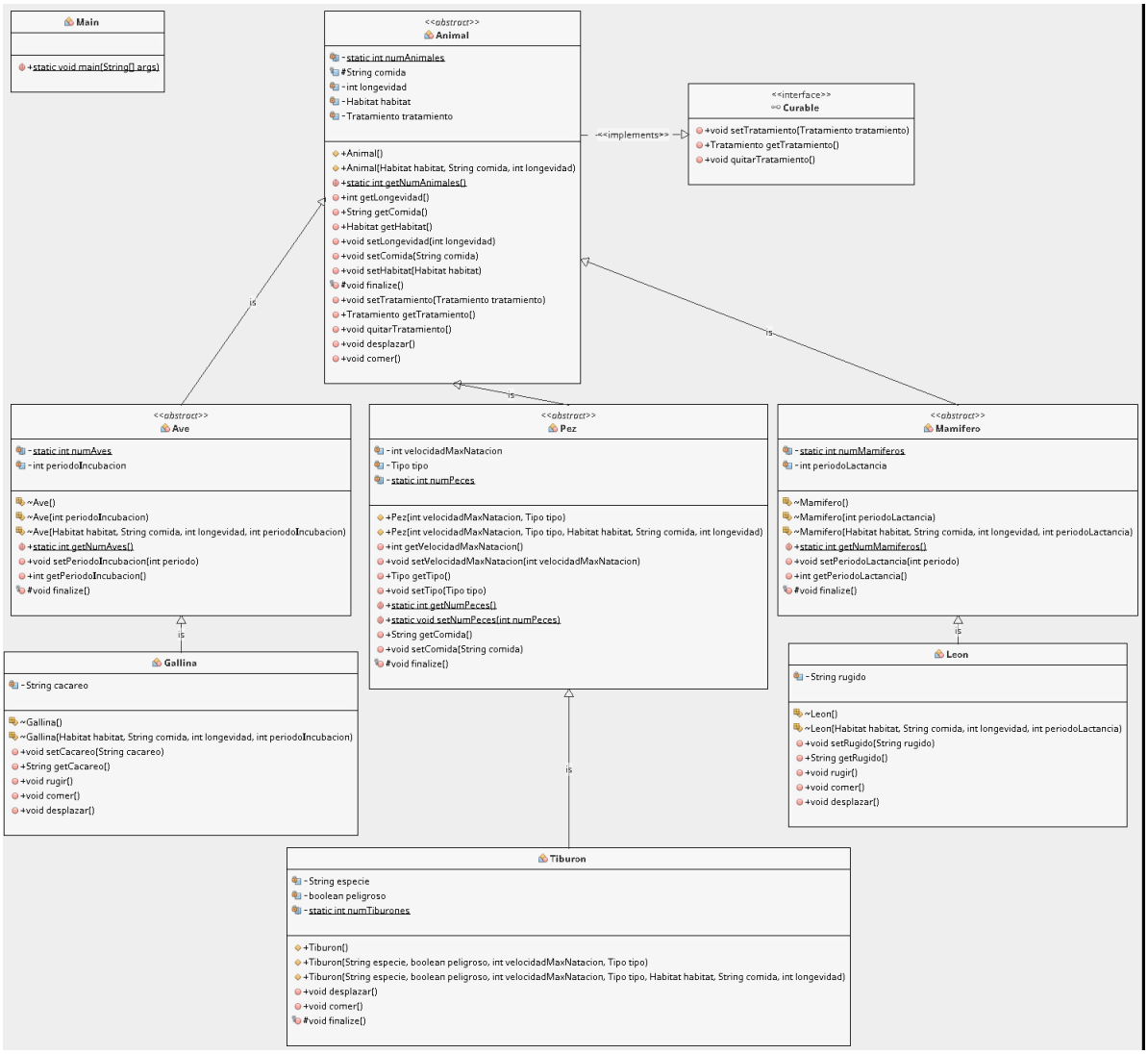
Videojuego



Colegio

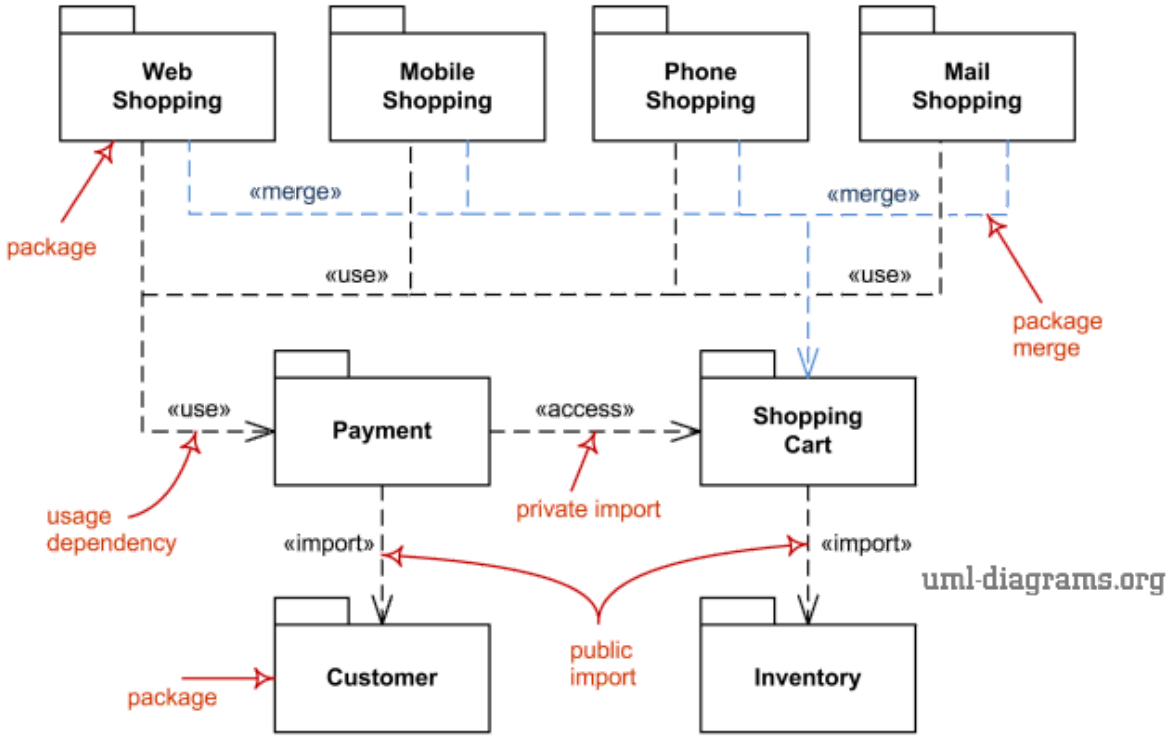

<<interface>>
Asignable
~void asignarTutoria(Escolarizable.Curso curso)
~void asignarJefatura(boolean asignada)
<<interface>>
Escolarizable
+void matricular(Curso curso)
+void desmatricular()
Main
+static void nuevoAlumno(Alumno alumno)
+static void main(String[] args)
1..1
<<implements>>
<<abstract>>
Profesor
- String nombre
- String apellidos
- GregorianCalendar nacimiento
- String[] asignaturas
- static int numProfesores
- boolean jefatura
- Escolarizable.Curso tutoria
+Profesor()
+Profesor(String nombre, String apellidos, GregorianCalendar nacimiento, String[] asignaturas, boolean jefatura, Escolarizable.Curso tutoria)
+String getNombre()
+void setNombre(String nombre)
+String getApellidos()
+void setApellidos(String apellidos)
+GregorianCalendar getNacimiento()
+void setNacimiento(GregorianCalendar nacimiento)
+String[] getAsignaturas()
+void setAsignaturas(String[] asignaturas)
+static int getNumProfesores()
+static void setNumProfesores(int numProfesores)
+boolean isJefatura()
+void setJefatura(boolean jefatura)
+Escolarizable.Curso getTutoria()
+void setTutoria(Escolarizable.Curso tutoria)
+void asignarTutoria(Escolarizable.Curso curso)
+void asignarJefatura(boolean asignada)
#void finalize()
<<abstract>>
Alumno
- String nombre
- String apellidos
- Date nacimiento
- int[] notas
- static int numAlumnos
#Curso curso
+Alumno()
+Alumno(String nombre, String apellidos)
+Alumno(String nombre, String apellidos, Date nacimiento, int[] notas)
+String getNombre()
+void setNombre(String nombre)
+String getApellidos()
+void setApellidos(String apellidos)
+Date getNacimiento()
+void setNacimiento(Date nacimiento)
+String getNotas_str()
+int[] getNotas()
+void setNotas(int[] notas)
+Curso getCurso()
+static int getNumAlumnos()
#void finalize()
is
AlumnoCiclo
+AlumnoCiclo()
+AlumnoCiclo(String nombre, String apellidos)
+AlumnoCiclo(String nombre, String apellidos, Date nacimiento, int[] notas)
+void matricular(Curso curso)
+void desmatricular()
AlumnoESO
- String nombrePadre
- String nombreMadre
+AlumnoESO()
+AlumnoESO(String nombre, String apellidos)
+AlumnoESO(String nombre, String apellidos, Date nacimiento, int[] notas)
+String getNombrePadre()
+void setNombrePadre(String nombrePadre)
+String getNombreMadre()
+void setNombreMadre(String nombreMadre)
+void matricular(Curso curso)
+void desmatricular()
ProfesorESO
- static int numProfesoresESO
+ProfesorESO()
+ProfesorESO(String nombre, String apellidos, GregorianCalendar nacimiento, String[] asignaturas, boolean jefatura, Escolarizable.Curso tutoria)
+static int getNumProfesoresESO()
+static void setNumProfesoresESO(int numProfesoresESO)
#void finalize()

Zoo



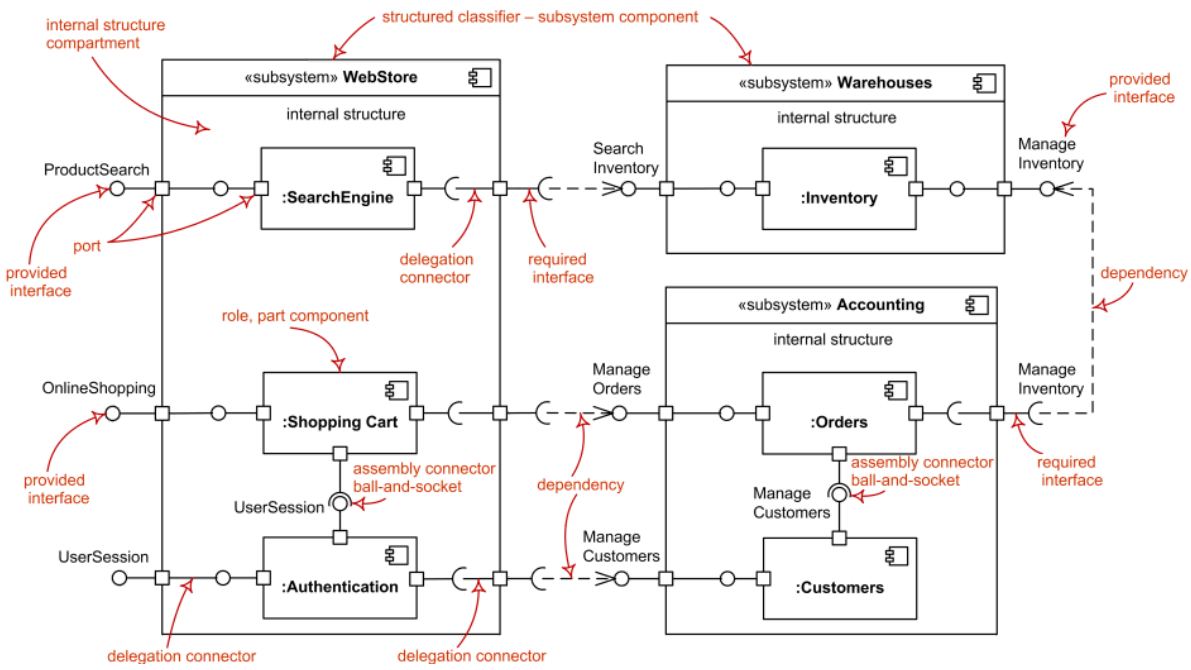
Diagramas de paquetes

- Un paquete es un espacio de nombres que se usa para agrupar elementos que están relacionados semánticamente.
- Si se elimina un paquete, también los elementos que pertenecen al paquete.
- Un paquete a su vez se puede empaquetar, por lo que cualquier paquete también podría ser miembro de otros paquetes.



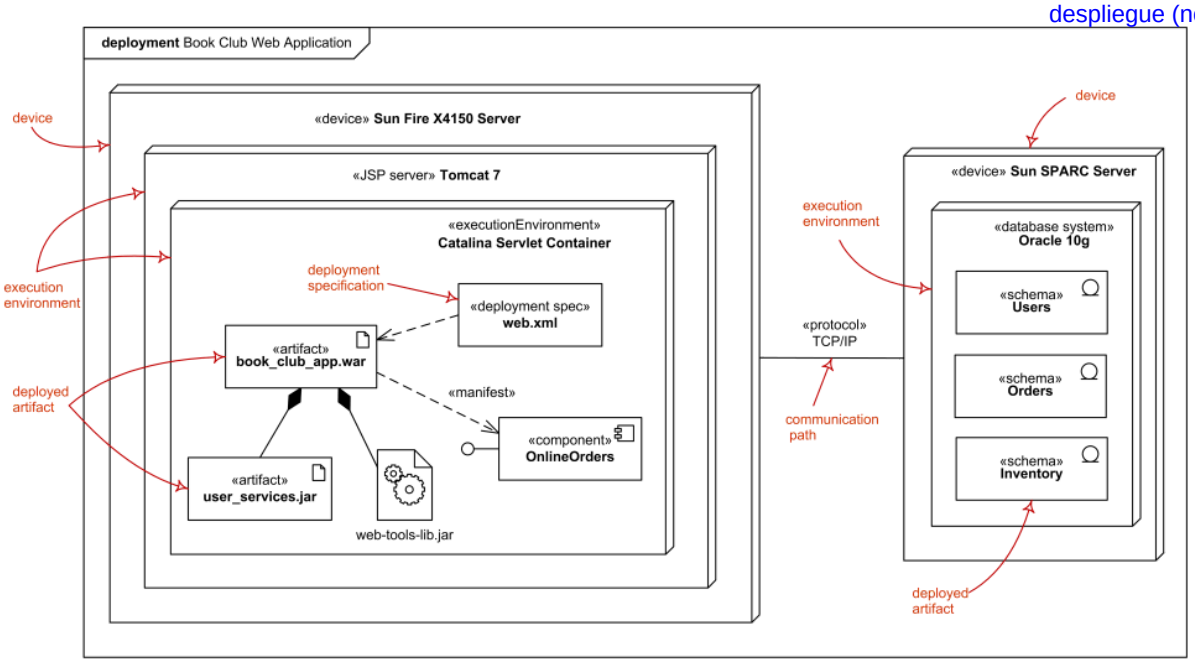
Diagramas de componentes

- Un componente es una clase que representa una parte modular de un sistema.



Diagramas de implementación (deployment)

- Muestra la arquitectura del sistema como despliegue (distribución) de artefactos de software a destinos de despliegue (nodos).



Software

- Enterprise Architect
- Visual Paradigm
- Microsoft Visio
- Dia, ArgoUML, Umbrello
- Plugins para Netbeans (**easyUML**, plantUML)
- Plugins para Eclipse (...)
- Plugins para IntellJ Idea (...)
- Lista exhaustiva

ARTEFACTOS

- Los artefactos representan elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo.
- Ejemplos de artefactos son archivos ejecutables, bibliotecas, archivos, esquemas de base de datos, archivos de configuración, etc.

NODOS (I)

- El destino de la implementación suele estar representado por un nodo que es un dispositivo de hardware o algún entorno de ejecución de software.

NODOS (II)

Un nodo puede contener en su interior otros nodos

NODOS (III)

- Los nodos pueden conectarse a través de vías de comunicación para crear sistemas en red de complejidad arbitraria.

MANIFESTACIÓN DE COMPONENTES

- En UML 2.x, los artefactos se implementan en los nodos, y los artefactos podrían manifestar (implementar) componentes. Los componentes se implementan en los nodos indirectamente a través de artefactos.