PUCE Virtual

CLASE

4

Programación orientada a objetos

Modelado orientado a objetos

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
sin interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

Hoy iniciaremos una parte muy importante de la asignatura relacionada con el diseño de aplicaciones orientadas a objetos. En esta clase analizaremos cómo pasar de la comprensión de un problema a una estructura de software organizada en clases, atributos, métodos y relaciones. Para lograrlo, estudiaremos primero el modelado orientado a objetos, donde aprenderemos a identificar los elementos del modelo dentro de un sistema, es decir, reconocer las clases y qué información o comportamiento debe representar cada una. Este proceso es esencial porque permite construir una representación clara del sistema antes de implementarlo en código.

Además, incorporaremos un aspecto clave del diseño orientado a objetos: el encapsulamiento y la visibilidad, que permiten controlar cómo se accede a los datos y comportamientos dentro de una clase. Esto implica definir qué atributos y métodos deben ser públicos, privados o protegidos, garantizando no solo una mejor organización del código, sino también mayor seguridad y control sobre la información, especialmente en sistemas donde se manejan datos sensibles. Posteriormente, profundizaremos en las relaciones entre clases, analizando conceptos fundamentales como asociación, agregación, composición y herencia, así como la forma correcta de representarlos mediante diagramas dentro de UML (Unified Modeling Language).

7. MODELADO ORIENTADO A OBJETOS

El modelado orientado a objetos es una etapa clave en el diseño de software, ya que permite transformar el análisis del problema en una estructura técnica compuesta por clases, objetos, atributos, métodos y relaciones, facilitando la representación organizada de las entidades del sistema mediante abstracciones que ayudan a manejar su complejidad (Oviedo Regino, 2015). Desde la perspectiva de la ingeniería de software, este modelado define la forma en que interactúan los componentes del sistema, sus responsabilidades y los mecanismos de comunicación entre ellos, lo que contribuye a mejorar la reutilización del código, la mantenibilidad y la extensibilidad del sistema (Blanco Fernández, 2020).
En el ámbito de la ciberseguridad, el modelado orientado a objetos también permite incorporar principios de seguridad desde el diseño, como el control de acceso, el principio de privilegio mínimo, la encapsulación de información sensible y la separación de responsabilidades, garantizando que cada objeto solo pueda ejecutar las operaciones que le corresponden (Hernández Bejarano, 2023).
Un ejemplo de aplicación es el diseño de un sistema de acceso seguro mediante clases como Usuario, Rol, Permiso y Recurso, donde el acceso a los recursos se gestiona a través de roles y permisos asociados. Este enfoque corresponde al modelo RBAC (Role-Based Access Control), que permite una administración más flexible de los accesos al asignar permisos a roles en lugar de hacerlo directamente a los usuarios (Phillips & Plott, 2021).

7.1 IDENTIFICACIÓN DE ELEMENTOS DEL MODELO

La identificación de elementos del modelo consiste en determinar qué entidades del dominio deben representarse como clases dentro del sistema, así como sus atributos, responsabilidades y relaciones.
La tabla 1 presenta la información organizada de los elementos de un sistema de control de acceso basado en roles, conocido como RBAC (Role-Based Access Control).

Tabla 1

Elementos del modelo de un sistema de control de acceso seguro

Clase	Descripción	Atributos posibles	Responsabilidades	Ejemplos / Observaciones
Usuario	Representa a la persona o entidad que interactúa con el sistema informático. En ciberseguridad contiene la información necesaria para identificar y autenticar a un individuo dentro del sistema.	– idUsuario – nombreUsuario – contraseñaHash – estadoCuenta – rolesAsignados	– Autenticarse en el sistema – Solicitar acceso a recursos – Consultar sus permisos	La contraseña no debe almacenarse en texto plano, sino como un hash criptográfico, lo cual es una práctica básica de seguridad para proteger credenciales.
Rol	Representa un conjunto de responsabilidades o funciones dentro del sistema. Permite agrupar permisos para facilitar la administración	– idRol – nombreRol – descripción – listaPermisos	– Agrupar permisos relacionados – Determinar las acciones que puede realizar un usuario	Ejemplos de roles: Administrador, Auditor, Usuario estándar. Permite aplicar el principio de separación de responsabilidades.

	del control de acceso.			
Permiso	Define las acciones específicas que pueden realizarse sobre un recurso del sistema.	– idPermiso – tipoOperacion – recursoAsociado	– Definir acciones permitidas – Controlar operaciones sobre recursos	Ejemplos: Leer archivo, Modificar base de datos, Eliminar registro, Ejecutar servicio. Permite aplicar el principio de privilegio mínimo.
Recurso	Representa cualquier elemento del sistema que requiere protección mediante mecanismos de control de acceso.	– idRecurso – nombreRecurso – tipoRecurso – nivelSeguridad	– Representar elementos protegidos del sistema – Permitir la aplicación de políticas de acceso	Ejemplos: Archivo, Base de datos, Servicio web, API del sistema.

Autor: Héctor Avalos Silva

Para reforzar el conocimiento adquirido, invito a revisar el contenido del siguiente video, donde a más de se muestra la forma de identificar los elementos de un modelo orientado a objetos.

- **Título**: Diagrama de clases
- **Descripción**: El video explica los fundamentos del UML (Unified Modeling Language) y cómo se utilizan los diagramas de clases para representar la estructura de un sistema. En particular, muestra cómo identificar y representar elementos como clases, atributos, métodos y relaciones, que son componentes esenciales del modelado orientado a objetos
- **Enlace**: https://youtu.be/Xgr4fcApcE0

7.2 ENCAPSULAMIENTO Y VISIBILIDAD

Encapsulamiento: En la programación orientada a objetos, el encapsulamiento es el principio que consiste en agrupar datos (atributos) y comportamientos (métodos) dentro de una clase, restringiendo el acceso directo a sus componentes internos. Su objetivo es proteger la integridad de los datos y controlar cómo se modifican o consultan.

Visibilidad: La visibilidad, define los niveles de acceso a esos atributos y métodos mediante modificadores como private, protected y public (Oviedo Regino, 2015).

Por ejemplo, en una clase Usuario, la contraseña no debe ser accesible directamente. En su lugar, se implementan métodos que controlan su uso, como la verificación:

```
public class Usuario {
   //aquí otros atributos de la clase
   private String passwordHash;

   public boolean verificarPassword(String input) {
      return passwordHash.equals(hash(input));
   }

   private String hash(String input) {
      // lógica de cifrado
      return input;
   }
}
```

Aquí, el atributo passwordHash está encapsulado como privado, evitando que otras clases accedan directamente a él, esta es una práctica recomendada para proteger datos sensibles (Oviedo Regino, 2015).

Los modificadores de visibilidad permiten implementar distintos niveles de acceso dentro del sistema:

- **Privado** (private): restringe el acceso únicamente a la propia clase. Es el nivel más seguro y debe utilizarse para proteger información crítica como contraseñas, tokens o claves. Este enfoque sigue el principio de ocultamiento de información, esencial en el diseño seguro (Blanco Fernández, 2020).
- **Protegido** (protected): permite el acceso dentro de la clase y sus subclases. Es útil cuando se requiere reutilizar lógica en estructuras de herencia, por ejemplo, métodos de validación que serán extendidos por diferentes tipos de autenticación:

```
protected boolean validarCredenciales(String usuario, String password) {
   // lógica común de validación
   return true;
}
```

Sin embargo, su uso debe ser controlado, ya que amplía el alcance del acceso (Gervais, 2025).

- **Público** (public): permite el acceso desde cualquier parte del sistema. Representa la interfaz visible, por lo que debe limitarse a los métodos estrictamente necesarios y siempre incluir validaciones. En un sistema de acceso seguro, estos métodos son el punto de entrada:

```
public boolean login(String usuario, String password) {
   return verificarPassword(password);
}
```

Este diseño asegura que todas las operaciones pasen por controles definidos, lo cual es una práctica clave para construir sistemas robustos y mantenibles (Phillips & Plott, 2021).

En una implementación, una clase como SistemaAutenticacion mantendría sus datos sensibles como privados, expondría únicamente métodos públicos para operaciones como autenticación, y utilizaría elementos protegidos solo cuando sea necesario compartir lógica entre clases relacionadas. Esto permite centralizar el control y reducir riesgos de seguridad.

En concreto, aplicar encapsulamiento correctamente implica seguir reglas claras, no exponer datos sensibles, validar siempre en los métodos públicos y aplicar el principio de mínimo privilegio. Un mal uso de la visibilidad puede comprometer el sistema completo, incluso si la lógica es correcta.

8. RELACIONES ENTRE CLASES

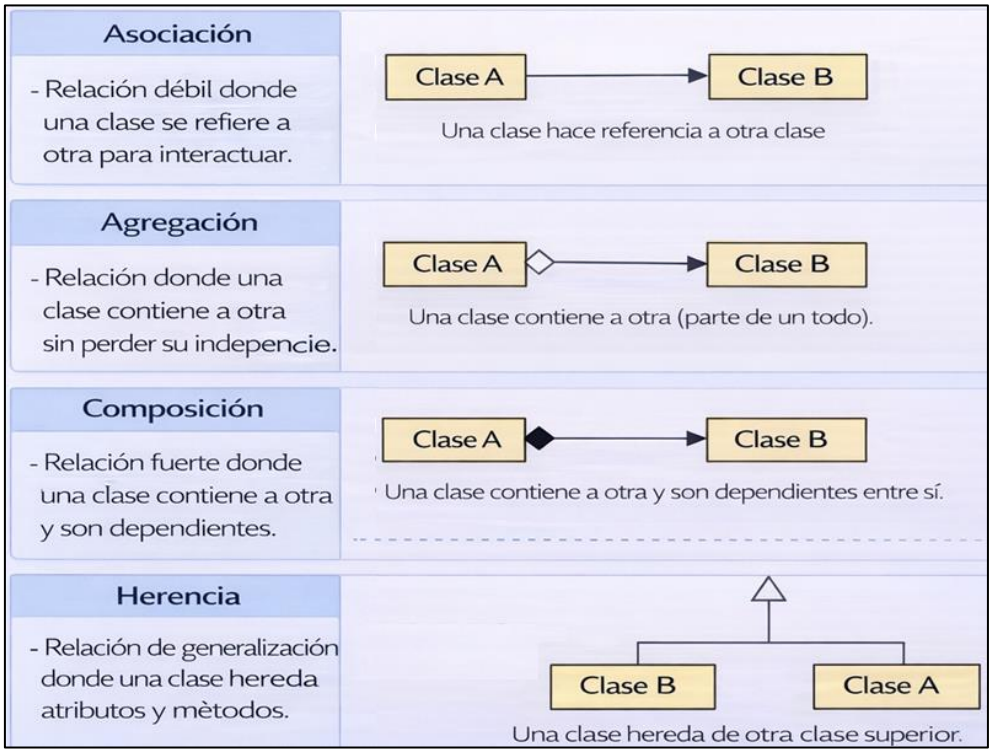
En el diseño de modelado orientado a objetos, uno de los aspectos más importantes es definir correctamente las relaciones entre las clases. Estas relaciones permiten representar cómo interactúan los objetos dentro del sistema y cómo se organiza la estructura lógica del software. En el contexto de ciberseguridad y control de acceso a sistemas, comprender estas relaciones es fundamental para diseñar arquitecturas seguras, mantenibles y desacopladas.

El modelado orientado a objetos no se limita a identificar clases, sino que incluye definir sus relaciones para representar correctamente el problema y estructurar la colaboración entre componentes, favoreciendo la reutilización del código (Oviedo Regino, 2015).

La figura 1 muestra los principales tipos de relaciones utilizadas en POO, junto con su representación habitual en los diagramas de clases.

Figura 1

Tipos de relaciones en programación orientada a objetos



Autor: Héctor Avalos Silva

A continuación, se desarrollan los principales tipos de relaciones entre clases aplicados al ejemplo de acceso seguro a sistemas, donde las entidades principales son: Usuario, Rol, Permiso, Recurso.

8.1 Asociación, agregación, composición, dependencia y herencia

a. Asociación

La asociación es una relación estructural básica entre clases que indica que los objetos de una clase están conectados o interactúan con objetos de otra clase. Esta relación representa una colaboración entre entidades dentro del sistema.

Según (Hernández Bejarano, 2023), una asociación describe una conexión lógica entre clases que permite a los objetos comunicarse o intercambiar información, sin implicar necesariamente dependencia fuerte o propiedad sobre el ciclo de vida de los objetos.

Ejemplo en control de acceso: En un sistema de ciberseguridad, un usuario tiene uno o varios roles. Esto significa que existe una relación entre la clase Usuario y la clase Rol.

Representación: Usuario ——tiene ——Rol

En términos del dominio del problema:

- Un usuario puede tener varios roles.
- Un rol puede ser asignado a múltiples usuarios.

Ejemplo:

Usuario	Rol
Ana	Administrador
Luis	Auditor
María	Usuario

Esta relación se denomina muchos a muchos, ya que varios usuarios pueden compartir un rol.

Justificación de diseño: La asociación permite modelar la asignación de roles de forma flexible. En sistemas de seguridad modernos se utiliza el modelo RBAC (Role-Based Access Control), donde los permisos no se asignan directamente a usuarios, sino a roles.

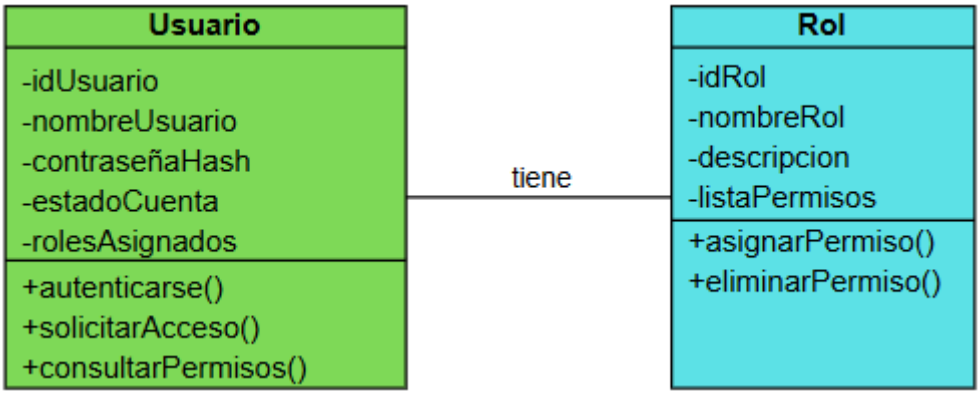
Esto mejora la administración del sistema porque evita duplicación de permisos, facilita la gestión de accesos, reduce errores de configuración.

Tal como indican (Phillips & Plott, 2021), separar responsabilidades mediante relaciones bien definidas mejora la mantenibilidad y escalabilidad del software.

La figura 2 muestra a las clases Usuario y Rol conectadas mediante una relación de asociación denominada “tiene”, lo que indica que un usuario puede tener asociados uno o varios roles dentro del sistema.

Figura 2

Relación de asociación entre las clases Usuario y Rol



Autor: Héctor Avalos Silva

b. Agregación

La agregación es un tipo especial de asociación que representa una relación todo–parte, donde una clase contiene a otras pero las partes pueden existir independientemente del todo.

De acuerdo con (Guardati Buemo, 2016), la agregación permite modelar estructuras jerárquicas donde un objeto agrupa a otros, sin que estos dependan completamente de su existencia.

Se representa en UML con un rombo vacío.

Ejemplo en control de acceso

Un rol contiene permisos.

Representación: Rol ◇ ——contiene ——— Permiso

Interpretación:

- Un rol agrupa varios permisos.
- Un permiso puede existir, aunque el rol sea eliminado.
- El mismo permiso puede pertenecer a distintos roles.

Ejemplo:

Rol	Permisos
Administrador	Crear usuario
Administrador	Eliminar usuario
Administrador	Modificar recurso
Auditor	Ver registros
Auditor	Consultar reportes

Justificación de diseño

Esta relación se modela como agregación porque:

- Los permisos son entidades reutilizables
- No dependen exclusivamente de un rol
- Pueden asignarse a múltiples roles

Por ejemplo:

El permiso “leer archivo” podría pertenecer tanto al rol Administrador como al rol Auditor.

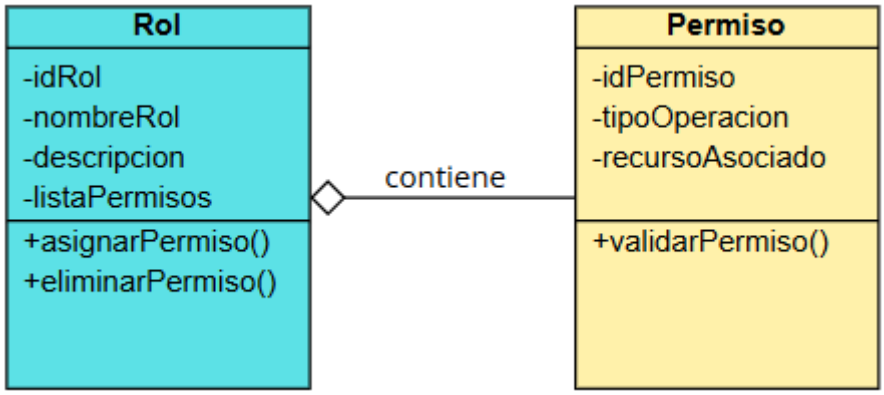
Este diseño es fundamental en sistemas de ciberseguridad porque permite reutilizar reglas de acceso y mantener un control centralizado sobre los permisos.

Según Gervais (s. f.), las relaciones de agregación favorecen la modularidad del sistema, permitiendo modificar partes del modelo sin afectar la estructura global.

La figura 3 muestra una relación de agregación que indica que un rol agrupa o contiene permisos, esta estructura permite organizar los privilegios del sistema de manera flexible y escalable, facilitando la administración del control de acceso mediante el modelo control de acceso basado en roles.

Figura 3

Relación de agregación entre Rol y Permiso



Autor: Héctor Avalos Silva

c. Composición

La composición es una relación fuerte de tipo todo–parte, donde el objeto contenido depende completamente del objeto contenedor. Esto significa que los componentes solo existen mientras existe el objeto compuesto y no pueden compartirse con otros objetos. Si el objeto principal se elimina, sus componentes también desaparecen.

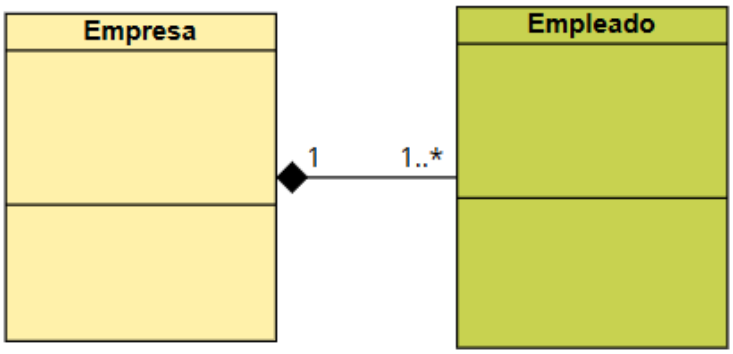
Guardati Buemo (2007) explica que en la composición existe una dependencia total del ciclo de vida, lo que significa que cuando el objeto contenedor se destruye, los objetos contenidos también desaparecen.

En UML (Unified Modeling Language), esta relación se representa con un rombo negro ubicado en el lado de la clase que representa el todo.

La figura 4 muestra un ejemplo de relación de composición entre la clase Empresa y la clase Empleado. Un objeto Empresa está a su vez compuesto por uno o varios objetos del tipo empleado, el tiempo de vida de los objetos Empleado depende del tiempo de vida de Empresa, ya que si no existe una Empresa no pueden existir sus empleados.

Figura 4

Relación de composición entre las clases Empresa y Empleado



Autor: Héctor Avalos Silva

d. Dependencia

La dependencia representa una relación donde una clase utiliza temporalmente a otra para realizar alguna operación.

Según Blanco Fernández (2020), la dependencia indica que un cambio en una clase podría afectar a otra que la utiliza.

No implica propiedad ni asociación permanente.

Ejemplo en control de acceso

El Usuario solicita acceso a un recurso, pero el acceso depende de los permisos asociados a su rol.

Flujo:

Usuario → consulta → Rol

Rol → verifica → Permiso

Permiso → autoriza → Recurso

Este flujo refleja la lógica típica de autorización en un sistema seguro.

Justificación de diseño:
Esta relación muestra el proceso de verificación de acceso.
Por ejemplo:

i. El usuario intenta acceder a un archivo.
ii. El sistema identifica los roles del usuario.
iii. Se revisan los permisos asociados a esos roles.
iv. Se determina si el acceso al recurso está autorizado.

Este proceso representa la interacción dinámica entre objetos dentro del sistema.
Plott y Phillips (2021) señalan que las dependencias permiten modelar colaboraciones entre objetos durante la ejecución del programa.

e. Herencia

La herencia representa una relación jerárquica donde una clase hereda atributos y métodos de otra.
Se conoce como relación "es-un" (is-a).
Ejemplo: EventoLoginFallido es un EventoSeguridad.
Representación en UML: **EventoLoginFallido** ——▷ **EventoSeguridad**
Línea continua con triángulo blanco.
Ejemplo en Java:
public class **EventoLoginFallido** extends **EventoSeguridad {**

}
Aquí:

- la clase hija hereda atributos y métodos
- existe una relación estructural fuerte.

Considerando el ejemplo del sistema de acceso seguro, en la tabla 2 se especifica, para cada par de clases el tipo de relación, la multiplicidad y la justificación desde la perspectiva de ciberseguridad. Se define la estructura del control de acceso, garantizando que los permisos no se asignen directamente a los usuarios, sino que se administren a través de roles, lo que facilita la gestión de privilegios, mejora la escalabilidad del sistema y contribuye a aplicar principios fundamentales de seguridad.

Tabla 2

Descripción de las relaciones entre las clases del sistema de acceso seguro.

Clase origen	Clase destino	Tipo de relación	Multiplicidad	Justificación en ciberseguridad
Usuario	**Rol**	**Asociación**	Un usuario puede tener uno o varios roles (1..**) y un rol puede estar asignado a varios usuarios (0..**)	Permite administrar permisos de manera eficiente sin asignarlos directamente a cada usuario.
Rol	**Permiso**	**Agregación**	Un rol puede contener varios permisos (1..**) y un permiso puede pertenecer a varios roles (0..**)	Los roles agrupan permisos relacionados para simplificar la gestión de seguridad.
Permiso	**Recurso**	**Asociación**	Un permiso se aplica sobre un recurso específico (1) y un recurso puede tener varios permisos asociados (0..*)	Define qué operaciones pueden realizarse sobre cada recurso protegido.
Usuario	**Recurso**	**Asociación indirecta**	La relación se establece a través de roles y permisos	Un usuario accede a recursos únicamente si posee un rol que incluya el permiso correspondiente.

Autor: Héctor Avalos Silva

Importancia de definir correctamente las relaciones

Definir correctamente las relaciones entre clases permite:

- representar fielmente el dominio del problema
- mejorar la seguridad del sistema
- facilitar la reutilización del código
- reducir el acoplamiento entre componentes

Además, un diseño adecuado permite aplicar principios de arquitectura segura, como:

- separación de responsabilidades
- control centralizado de permisos
- escalabilidad del sistema

Oviedo Regino (2015) destaca que el modelado orientado a objetos bien estructurado permite traducir correctamente los requerimientos del sistema en estructuras de software coherentes y mantenibles.

8.2 REPRESENTACIÓN GRÁFICA DEL DISEÑO

Una vez identificadas las clases del modelo orientado a objetos, es necesario representarlas de forma gráfica para facilitar la comprensión del diseño. Para ello se utiliza el Lenguaje Unificado de Modelado (UML), el cual proporciona diagramas que permiten visualizar la estructura del sistema antes de su implementación (Gervais, 2025).

El diagrama de clases UML es uno de los más utilizados en el diseño orientado a objetos, ya que muestra:

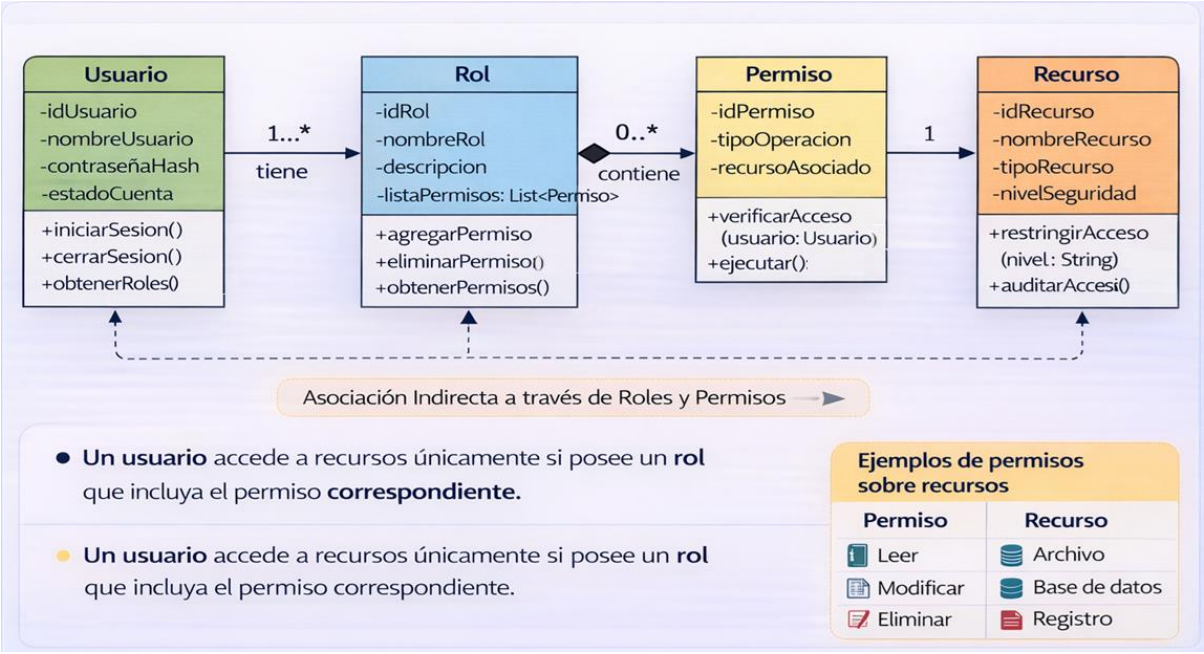
- Las clases del sistema
- Sus atributos
- Sus métodos
- Las relaciones entre clases

Según la literatura sobre programación orientada a objetos, la representación gráfica permite validar el diseño antes de la fase de programación, evitando errores estructurales en la arquitectura del software (Guardati Buemo, 2016).

En la figura 5 se observa los diferentes tipos de relaciones identificadas entre clases del sistema de control de acceso seguro.

Figura 5

Relaciones entre clase del sistema de control de acceso seguro



Autor: Héctor Avalos Silva

Este diagrama representa el flujo lógico de autorización dentro del sistema.

El proceso funciona de la siguiente manera:

1. Un usuario intenta acceder a un recurso.
2. El sistema consulta los roles asignados al usuario.
3. Cada rol contiene una lista de permisos.
4. El permiso define si una operación específica puede realizarse sobre un recurso protegido.

Este modelo permite implementar mecanismos de seguridad como:

- Autorización basada en roles
- Control de acceso granular
- Auditoría de permisos

Con el propósito de reforzar los conocimientos sobre relaciones entre clase y su representación gráfica, ingresa al siguiente video.

- **Título:** Relaciones entre clases
- **Descripción:** En el video se describen las principales relaciones entre clases que se utilizan en el diseño orientado a objetos, estas relaciones se representan gráficamente en diagramas de clases UML, que permiten visualizar la estructura del sistema y las interacciones entre las clases mediante líneas, flechas y símbolos específicos dentro del diagrama.
- **Enlace:** https://youtu.be/qVoeMHMG4wc

Glosario:

Blanco Fernández, Y. (2020). *Introducción a la programación orientada a objetos.* Andavira.

Gervais, L. (2025). *Aprender Programación Orientada a Objetos con Java.* (eni, Ed.) https://doi.org/978-2-409-05028-2

Guardati Buemo, S. (2016). *Estructuras de datos básicas: Programación orientada a objetos con Java.* Alfaomega.

Hernández Bejarano, M. &. (2023). *Programación Orientada a Objetos en Java: Buenas prácticas. Ingeniería de sistemas.* https://doi.org/978-958-792-463-3

Oviedo Regino, E. M. (2015). *Lógica de programación orientada a objetos.* Ecoe Ediciones. https://doi.org/978-958-711-136-3

Phillips, D., & Plott, S. F. (2021). *Python Object-Oriented Programming: Build robust and maintainable object-oriented Python applications and libraries* (4ta ed.). Packt Publishing Limited. https://doi.org/978-1801077262

DEFINICIÓN DE TÉRMINOS

- **Modelado orientado a objetos:** Es el proceso de análisis y diseño mediante el cual se representa un sistema de software utilizando clases, objetos, atributos, métodos y relaciones, con el propósito de reflejar las entidades y comportamientos del dominio del problema. Este modelado permite abstraer la realidad en componentes organizados que facilitan la comprensión, construcción y mantenimiento del sistema, sirviendo como puente entre el análisis del problema y su implementación en código.
- **Relaciones entre clase:** Describen la forma en que las clases de un sistema interactúan o dependen unas de otras dentro de un modelo orientado a objetos. Estas relaciones permiten representar cómo los objetos colaboran para cumplir las funcionalidades del sistema y pueden adoptar diferentes formas, como asociación, agregación, composición y

herencia. En los diagramas de clases se utilizan para mostrar la estructura del sistema y las dependencias existentes entre sus componentes.

RECURSOS DE PROFUNDIZACIÓN

Recurso 1:

- **Título:** Modelado OO de un Sistema de Detección de Intrusiones (SDI)
- **Descripción:** El modelo OO del SDI identifica los elementos que conforman este sistema, estableciendo clases, atributos, comportamiento y relaciones pertinentes.
- **Documento:** Recurso de profundización 1 Ejemplo guiado de modelado OO SDI.docx

Recurso 2:

- **Título:** Representación gráfica en UML de las relaciones entre clases y su implementación en lenguaje Java de un sistema de detección de intrusos (SDI).
- **Descripción:** Este recurso muestra de forma gráfica a los elementos del MOO y sus relaciones a través de UML, también implementa en lenguaje Java el modelo del SDI.
- **Documento:** Recurso de profundización 2 Gráfico UML e implementación en Java modelo OO SDI.docx

ACTIVIDAD DE EVALUACION

PREGUNTAS DE AUTOEVALUACION

<table>
<tr><td>INSTRUCCIONES</td><td colspan="6">Lea cuidadosamente cada situación planteada y seleccione la alternativa correcta. Las preguntas se basan en escenarios de sistemas de seguridad informática, donde se aplican conceptos de modelado orientado a objetos, identificación de elementos del modelo y relaciones entre clases.

Después de seleccionar su respuesta, revise la retroalimentación para comprender la justificación de la respuesta correcta</td></tr>
<tr><td>Titulo</td><td colspan="6">Autoevaluación: Modelado Orientado a Objetos aplicado a Sistemas de Ciberseguridad</td></tr>
<tr><td>Dificultad</td><td></td><td>Baja</td><td>x</td><td>Media</td><td></td><td>Alta</td></tr>
<tr><td rowspan="3">RDAs Evaluados:</td><td></td><td colspan="5">RDA 1</td></tr>
<tr><td>x</td><td colspan="5">RDA 2</td></tr>
<tr><td></td><td colspan="5">RDA 3</td></tr>
<tr><td>Secciones evaluadas</td><td></td><td colspan="5">7.2 Modelado orientado a objetos

7.2.1 Identificación de elementos del modelo

7.2.2 Representación gráfica del diseño

8 Relaciones entre clases

8.1 Asociación, agregación, composición, dependencia

8.2 Herencia en el modelo</td></tr>
</table>

PREGUNTA 1	Una organización está desarrollando un sistema de monitoreo de seguridad que registra eventos como intentos de acceso fallidos, accesos a recursos restringidos y actividad sospechosa en la red. Durante la fase de diseño del sistema, los ingenieros de software identifican clases como Usuario, EventoSeguridad, AlertaSeguridad y RecursoProtegido para representar los componentes del sistema. En el contexto del modelado orientado a objetos, ¿cuál es el propósito principal de identificar estas clases durante el diseño del sistema?
CORRECTA	Representar las entidades principales del dominio de seguridad del sistema mediante clases que encapsulan atributos y comportamientos.
Incorrecta 1	Determinar únicamente la configuración del firewall que protegerá el sistema.
Incorrecta 2	Definir exclusivamente la estructura de la base de datos donde se almacenarán los registros.
Incorrecta 3	Establecer únicamente los algoritmos de cifrado utilizados en el sistema.
Retroalimentación de las respuestas	
CORRECTA	La identificación de clases es un paso fundamental del modelado orientado a objetos, ya que permite representar las entidades relevantes del dominio del problema (por ejemplo, usuarios, eventos o alertas de seguridad) mediante estructuras que contienen atributos y métodos. Esto facilita la organización del sistema antes de su implementación (Oviedo Regino, 2015).
Incorrecta 1	La configuración de un firewall corresponde a la administración de infraestructura de seguridad, no al modelado de software.
Incorrecta 2	La estructura de la base de datos es parte del diseño de datos, mientras que el modelado orientado a objetos describe la arquitectura lógica del sistema.
Incorrecta 3	Los algoritmos de cifrado son mecanismos de seguridad, pero no representan el objetivo principal del modelado orientado a objetos.
PREGUNTA 2	En un sistema de control de accesos utilizado en ciberseguridad, se modelan las clases Usuario y Rol. Cada usuario del sistema puede tener uno o varios roles (por ejemplo: administrador, analista de seguridad o auditor), y cada rol puede estar asignado a varios usuarios. ¿Qué tipo de relación representa mejor esta interacción entre las clases en un diagrama de clases de UML (Unified Modeling Language)?
CORRECTA	Asociación

Incorrecta 1	Herencia
Incorrecta 2	Composición
Incorrecta 3	Dependencia
Retroalimentación de las respuestas	
CORRECTA	La asociación representa una relación estructural donde dos clases se relacionan conceptualmente, como ocurre entre usuarios y roles en sistemas de control de acceso basados en roles. Esta relación permite que un usuario tenga múltiples roles y que un rol sea compartido por varios usuarios (Blanco Fernández, 2020).
Incorrecta 1	La herencia implica una relación jerárquica donde una clase hereda atributos y métodos de otra, lo cual no aplica en este caso.
Incorrecta 2	La composición implica una relación fuerte de dependencia del ciclo de vida entre objetos, lo cual no ocurre entre usuarios y roles.
Incorrecta 3	La dependencia representa un uso temporal entre clases y no una relación estructural permanente.
PREGUNTA 3	En un sistema de detección de intrusiones, una clase llamada AlertaSeguridad se genera cuando se detecta un evento sospechoso en el sistema. Cada alerta puede estar asociada a uno o varios EventosSeguridad que originaron la alerta. Si los diseñadores del sistema consideran que los eventos asociados a una alerta solo tienen sentido dentro del contexto de esa alerta, ¿qué tipo de relación sería más adecuada en el modelo orientado a objetos?
CORRECTA	Composición
Incorrecta 1	Dependencia
Incorrecta 2	Asociación
Incorrecta 3	Herencia
Retroalimentación de las respuestas	
CORRECTA	La composición representa una relación fuerte entre un objeto contenedor y sus partes, donde las partes dependen conceptualmente del todo. En este caso, los eventos asociados forman parte de la alerta y se interpretan dentro de su contexto de seguridad (Hernández Bejarano, 2023).
Incorrecta 1	La dependencia representa un uso temporal entre clases, lo cual no describe una relación estructural permanente.

Incorrecta 2	La asociación es una relación más general y no implica dependencia del ciclo de vida entre objetos.
Incorrecta 3	La asociación es una relación más general y no implica dependencia del ciclo de vida entre objetos.
PREGUNTA 4	Durante el diseño de un sistema de monitoreo de seguridad, se modela una clase general llamada EventoSeguridad, que representa cualquier tipo de evento registrado por el sistema. Posteriormente se crean clases más específicas como EventoLoginFallido y EventoAccesoNoAutorizado, que comparten características comunes del evento general pero agregan comportamientos específicos. ¿Qué tipo de relación del modelo orientado a objetos se está aplicando en este diseño?
CORRECTA	Herencia
Incorrecta 1	Dependencia
Incorrecta 2	Agregación
Incorrecta 3	Asociación
Retroalimentación de las respuestas	
CORRECTA	La herencia permite crear clases especializadas que heredan atributos y métodos de una clase más general. En este caso, las clases específicas de eventos heredan características de la clase base EventoSeguridad, lo que facilita la reutilización del código y la organización jerárquica del sistema (Blanco Fernández, 2020).
Incorrecta 1	La dependencia representa una relación de uso temporal entre clases.
Incorrecta 2	La agregación describe una relación todo-parte débil entre objetos.
Incorrecta 3	La asociación describe una relación general entre clases sin jerarquía.

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec