|  |  |
| --- | --- |
| Programa Académico | **Ingeniería de Sistemas** |
| Unidad Académica | **Ingeniería de Sistemas** |
| Asignatura | **Programación Orientada A Objetos** |
| Código | **IS0207** |
| Semestre | **02** |
| Créditos Académicos | **3** |
| Tipo Asignatura | **Teórico - Práctica** |
| Ciclo de Formación | **Ciclo de Fundamentación** |
| Componente de formación | **Desarrollo de Software** |
| Modalidad | **Presencial** |
| Descripción de la asignatura | La asignatura de Programación Orientada a Objetos (POO) es un componente crucial en el programa de ingeniería de sistemas, diseñado para introducir a los estudiantes en los principios y prácticas de la programación utilizando el paradigma de la orientación a objetos. Este curso suele abarcar varios conceptos fundamentales y técnicas esenciales para resolver problemas, modelar y desarrollar software de manera eficiente y efectiva; Este conjunto de conceptos, relaciones y principios forma la base de la Programación Orientada a Objetos y está diseñado para fomentar el desarrollo de software más organizado, mantenible y escalable. |

|  |  |
| --- | --- |
| Control de Versiones | **1.0** |
| Preparo | **Manuel Alberto González González** |
| Fecha |  |
| Acta Comité Curricular |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resultados de Aprendizaje de Asignatura | | | | |
| Código | Dimensión | Dominio Cognitivo | RAP | Descripción |
| **RAA-1** | **Hacer** | Recordar |  | Equipar a los estudiantes con conocimientos esenciales en el uso del paradigma de la orientación a objetos |
| **RAA-2** | **SABER** | Entender |  | proporcionar a los estudiantes habilidades prácticas en el paradigma de programación orientada a objetos |
| **RAA-3** | **SABER - HACER** | Aplicar |  | Aplicar principios y técnicas en el desarrollo de software real, preparándolos para enfrentar desafíos en entornos profesionales en el futuro. |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAA | Temas | Resultado de Aprendizaje Especifico | | Rúbrica | | | | | | | |
| Peso % | Criterios de evaluación | Niveles de desempeño | | | | |
| Id. | Descripción |  |  | Superior | Alto | Básico | Bajo |
| RAA-1 | Abstracción y Propiedades de la abstracción; representación en UML | RAE-1 | Comprender y utilizar de manera efectiva los principios de la programación orientada a objetos, incluidos la abstracción, la encapsulación, la herencia, y el polimorfismo. |  | la habilidad del estudiante para diseñar e implementar clases que encapsulan datos y comportamientos de manera efectiva, reflejando una comprensión profunda de la abstracción  Mide la capacidad del estudiante para promover la reutilización del código mediante la implementación efectiva de abstracción.  Habilidad del estudiante para representar abstracciones en diagramas UML, mostrando una comprensión de cómo visualizar y comunicar diseños de software abstraídos. | El estudiante demuestra una comprensión excepcional de la abstracción, diseñando clases con una clara separación de responsabilidades y atributos y métodos bien definidos que encapsulan perfectamente el comportamiento y estado esperado.  Implementa objetos de estas clases de manera que aprovecha al máximo sus capacidades, demostrando un entendimiento profundo de cómo la abstracción facilita un diseño de software modular y reutilizable.  Representa de manera precisa y detallada las clases y objetos en UML, incluyendo relaciones, visibilidad de atributos y métodos, y notación correcta. | El estudiante muestra una sólida comprensión de la abstracción, creando clases bien estructuradas y objetos que cumplen con su propósito dentro del diseño del software.  Utiliza la abstracción para mejorar la claridad y mantenibilidad del código, aunque con pequeñas áreas para mejorar en términos de optimización y eficiencia.  Representa las clases y objetos en UML de manera correcta, con algunos detalles menores que podrían mejorar para reflejar completamente las relaciones y características. | El estudiante tiene una comprensión básica de la abstracción, pero la implementación de clases y objetos muestra áreas de ambigüedad o una falta de cohesión y coherencia en el diseño.  La abstracción es utilizada, pero no se aprovecha plenamente para facilitar la reutilización del código y la reducción de la complejidad.  La representación en UML de clases y objetos es generalmente correcta pero falta detalle o precisión en la notación, mostrando una comprensión incompleta de las relaciones y la visibilidad. | El estudiante muestra una comprensión limitada de la abstracción, con clases y objetos mal definidos que no encapsulan adecuadamente el estado o el comportamiento, o que no se justifican dentro del contexto del diseño.  Hay un uso ineficaz o incorrecto de la abstracción, lo que resulta en un código confuso, difícil de mantener o reutilizar.  La representación en UML es inexacta o incompleta, con errores significativos en la notación, falta de relaciones clave, o una mala interpretación de la visibilidad de atributos y métodos. |
| Relaciones entre clases: Dependencia , Asociación, Agregación, Composición, Herencia | RAE-2 | Comprender y aplicar las relaciones entre objetos, incluyendo asociación, dependencia, agregación, composición, y herencia, para construir sistemas complejos y eficientes. |  | La capacidad del estudiante para identificar y aplicar relaciones de dependencia entre clases, donde un cambio en una clase puede afectar a otra.  Mide cómo el estudiante comprende y utiliza la asociación, una relación flexible que permite a las clases conocerse y comunicarse entre sí.  La habilidad del estudiante para implementar agregación y composición, representando relaciones "todo-parte" con distintos grados de dependencia.  La capacidad del estudiante para aplicar herencia, facilitando la reutilización de código y la creación de jerarquías de clases.  La habilidad del estudiante para representar visualmente relaciones entre clases utilizando diagramas UML. | Identifica y aplica correctamente las relaciones de dependencia en situaciones complejas, demostrando una comprensión profunda de cómo un cambio en una clase puede afectar a otra.  Diseña e implementa relaciones de asociación de manera excepcional, demostrando una comprensión avanzada de cómo las clases pueden cooperar sin estar fuertemente acopladas.  Modela relaciones de agregación de manera experta, entendiendo perfectamente la relación "todo-parte" donde las partes pueden existir independientemente del todo.  Aplica la composición de manera excepcional, demostrando un entendimiento avanzado de la relación "todo-parte" donde las partes no pueden existir sin el todo.  Demuestra una comprensión excepcional de la herencia, aplicándola de manera innovadora para reutilizar y extender el código eficientemente. | Reconoce y utiliza las relaciones de dependencia de manera efectiva, con una comprensión clara de su impacto en el diseño del software.  Implementa relaciones de asociación correctamente, mostrando una buena comprensión de la cooperación entre clases.  Utiliza la agregación de manera efectiva, con una comprensión clara de su significado y aplicación.  Implementa correctamente relaciones de composición, mostrando una buena comprensión de su importancia en el diseño de software.  Utiliza la herencia correctamente, mostrando una sólida comprensión de cómo puede facilitar la reutilización del código y la creación de jerarquías de clases. | Tiene una comprensión básica de las relaciones de dependencia, pero puede no identificar todas las situaciones pertinentes o aplicarlas de manera óptima.  Aplica relaciones de asociación con una comprensión básica, pero puede confundir con otros tipos de relaciones o no optimizar su uso.  Tiene una comprensión básica de la agregación, pero puede no aplicarla de manera consistente o confundirla con la composición.  Entiende la composición a un nivel básico, pero puede aplicarla de manera inconsistente o confundirla con la agregación.  Aplica la herencia con una comprensión básica, pero puede no aprovechar plenamente sus beneficios o confundirla con otros tipos de relaciones | Muestra una comprensión limitada o incorrecta de las relaciones de dependencia, con dificultades para identificarlas o aplicarlas adecuadamente.  Tiene dificultades para entender y aplicar relaciones de asociación, confundiéndolas con otros tipos de relaciones o aplicándolas incorrectamente.  Muestra una comprensión limitada de la agregación, con errores significativos en su aplicación o una completa confusión con la composición.  Tiene una comprensión pobre de la composición, con dificultades para distinguirla de la agregación o aplicarla correctamente.  Muestra una comprensión limitada de la herencia, con dificultades significativas para aplicarla de manera efectiva o entender su propósito. |
| Manejo de estructura de colección | RAE-3 | Utilizar estructuras de datos y colecciones para gestionar eficientemente conjuntos de objetos, aplicando conceptos de manejo de colecciones, como listas, conjuntos, y mapas. |  | La habilidad del estudiante para seleccionar la estructura de colección más adecuada para un problema dado, teniendo en cuenta factores como el tipo de datos, la necesidad de acceso ordenado, la unicidad de los elementos y el rendimiento.  Mide la capacidad del estudiante para implementar de manera eficiente operaciones comunes con colecciones, como inserción, eliminación, búsqueda y recorrido de elementos.  La habilidad del estudiante para utilizar estructuras de colección de manera creativa y efectiva para resolver problemas complejos de programación. | Implementa estructuras de colección personalizadas cuando es necesario, aprovecha métodos avanzados para filtrar, ordenar, y transformar datos eficientemente, y aplica patrones de diseño para mejorar la gestión de colecciones. | Utiliza estructuras de colección estándar (como listas, mapas, y conjuntos) de manera efectiva, emplea iteradores y bucles para manipular colecciones, y es capaz de realizar operaciones básicas de filtrado y ordenamiento. | Aplica estructuras de colección para almacenar y acceder a datos, pero con limitada consideración sobre la eficiencia o la idoneidad de la estructura elegida. Realiza operaciones simples de adición y eliminación, pero enfrenta dificultades con operaciones más complejas. | Utiliza de manera incorrecta las estructuras de colección, como seleccionar un tipo de colección inapropiado para la tarea o aplicar operaciones de manera que comprometa el rendimiento o la lógica del programa. Muestra dificultades significativas en la manipulación de colecciones para tareas básicas. |
| RAA-2 | Encapsulación, ocultación de información, operadores de accesibilidad | RAE-1 | Aplicar correctamente los operadores de accesibilidad para encapsular y proteger los datos dentro de una clase, promoviendo la seguridad y la integridad del software. |  | La habilidad del estudiante para utilizar modificadores de acceso (público, protegido, privado) de manera efectiva para implementar la encapsulación en sus diseños de clases.  La capacidad del estudiante para diseñar clases que implementen adecuadamente la encapsulación, protegiendo los datos y exponiendo solo lo necesario. | El estudiante demuestra una comprensión excepcional de la encapsulación, aplicándola de manera innovadora para asegurar una alta cohesión y bajo acoplamiento en el diseño de software. Implementa métodos de acceso (getters) y modificación (setters) de manera eficaz, garantizando la integridad de los datos. Diseña clases donde la exposición de los datos se controla rigurosamente, utilizando adecuadamente los modificadores de acceso para proteger los atributos y asegurar que las modificaciones de estado se realicen solo a través de métodos bien definidos. | El estudiante muestra una sólida comprensión y aplicación de la encapsulación, utilizando correctamente los modificadores de acceso y definiendo métodos de acceso y modificación cuando es apropiado. Las clases diseñadas mantienen un buen nivel de protección de los datos, aunque podrían beneficiarse de optimizaciones adicionales para mejorar la cohesión y reducir el acoplamiento. | El estudiante tiene una comprensión básica de la encapsulación, pero su aplicación es inconsistente o presenta errores en la selección de los modificadores de acceso adecuados. Puede haber una tendencia a sobreexponer datos de clase o a implementar un control insuficiente sobre la modificación de los datos. Aunque se utilizan métodos de acceso y modificación, su implementación no optimiza las ventajas de la encapsulación. | El estudiante muestra una comprensión limitada de la encapsulación, con una aplicación deficiente que resulta en un acoplamiento alto y una cohesión baja en el diseño de software. Hay un uso inadecuado o ausencia de modificadores de acceso, y los datos de clase están frecuentemente expuestos o modificados directamente, sin el control adecuado a través de métodos. |
| Clases abstractas, interfaces: métodos abstractos | RAE-2 | Distinguir entre clases abstractas e interfaces y saber cuándo y cómo utilizar cada una para diseñar sistemas flexibles y desacoplados. |  | La habilidad del estudiante para diseñar e implementar clases abstractas que proporcionen una base común para sus subclases.  la capacidad del estudiante para diseñar e implementar interfaces que especifiquen un conjunto de métodos que las clases de implementación deben proveer.  La habilidad del estudiante para utilizar clases abstractas e interfaces para resolver problemas de diseño de software. | El estudiante demuestra una comprensión excepcional y una aplicación innovadora de clases abstractas e interfaces. Diseña clases abstractas e interfaces que encapsulan de manera efectiva los conceptos de alto nivel, facilitando la extensión y la implementación en clases concretas. Aprovecha las fortalezas de ambos conceptos para maximizar la reutilización del código y la flexibilidad del diseño, implementando patrones de diseño que dependen de la abstracción efectiva. | El estudiante muestra una sólida comprensión y aplicación de clases abstractas e interfaces. Utiliza estas herramientas de abstracción para definir contratos claros y estructuras base, permitiendo una implementación coherente en clases derivadas. Hay un buen uso de la herencia y la implementación para promover la reutilización del código y la claridad en el diseño del software. | El estudiante tiene una comprensión básica de clases abstractas e interfaces, pero su aplicación puede ser inconsistente o presentar malentendidos en cuanto a cuándo y cómo usar cada uno. Las clases abstractas e interfaces definidas pueden carecer de una estructuración óptima, y la implementación en clases concretas no siempre aprovecha la flexibilidad y reutilización del código que estos conceptos ofrecen. | El estudiante muestra una comprensión limitada y aplicación inadecuada de clases abstractas e interfaces. Hay una falta significativa de comprensión de los conceptos, resultando en un diseño de software que no se beneficia de la abstracción, la reutilización del código, o la flexibilidad. Las clases abstractas e interfaces pueden ser utilizadas de manera incorrecta, o no se utilizan cuando su aplicación sería beneficiosa. |
| Rediseño de métodos (sobre escritura de métodos), Polimorfismo con herencia, Polimorfismo con interfaces | RAE-3 | Implementar polimorfismo a través de la herencia y las interfaces, permitiendo que el software sea más flexible y extensible |  | Mide la habilidad del estudiante para implementar polimorfismo a través de la herencia, utilizando métodos sobrescritos y clases base.  La capacidad del estudiante para implementar polimorfismo utilizando interfaces, permitiendo que clases no relacionadas implementen los mismos métodos. | El estudiante demuestra una comprensión excepcional y una aplicación innovadora del polimorfismo, tanto con herencia como con interfaces. Diseña sistemas de software que aprovechan de manera óptima el polimorfismo para mejorar la flexibilidad, la reutilización del código y la mantenibilidad. Implementa de forma creativa el polimorfismo para resolver problemas complejos de diseño, facilitando la extensión y modificación del comportamiento de las clases sin alterar las interfaces existentes. | El estudiante muestra una sólida comprensión y aplicación efectiva del polimorfismo, utilizando herencia e interfaces para permitir que los objetos sean tratados de manera uniforme. Aplica correctamente el polimorfismo para diseñar clases que son fácilmente extensibles y cuyo comportamiento puede modificarse o ampliarse sin afectar el código existente. | El estudiante tiene una comprensión básica del polimorfismo y puede aplicarlo en situaciones simples utilizando herencia o interfaces. Sin embargo, la implementación puede ser básica y no aprovechar plenamente las capacidades del polimorfismo para mejorar la flexibilidad y la reutilización del código. La aplicación puede carecer de sofisticación o puede no utilizar el polimorfismo donde sería más beneficioso. | El estudiante muestra una comprensión limitada y una aplicación inadecuada del polimorfismo. Hay una falta significativa de comprensión de cómo y cuándo aplicar el polimorfismo mediante herencia e interfaces, resultando en un diseño de software rígido, difícil de extender y mantener. El estudiante puede no aplicar el polimorfismo donde es necesario o hacerlo de manera que no contribuya a la claridad o eficiencia del código. |
| RAA-3 | Principios SOLID: Ligadura estática y dinámica, Principio de responsabilidad única, Principio cerrado abierto, Principio de inversión de dependencia, Principio de segregación de interfaz y Principio de sustitución Liskov | RAE-1 | Aplicar principios de diseño de software, como el Principio de Responsabilidad Única, Principio Abierto/Cerrado, Principio de Inversión de Dependencia, Principio de Segregación de Interfaz, y Principio de Sustitución de Liskov, para crear software bien estructurado, mantenible y escalable. |  | La capacidad del estudiante para diseñar clases que tengan una única responsabilidad y razón para cambiar.  Mide la habilidad del estudiante para diseñar software que sea extensible sin necesidad de modificar el código existente.  La habilidad del estudiante para diseñar subclases de manera que puedan ser usadas en lugar de una clase base sin alterar el comportamiento correcto del programa.  Mide la capacidad del estudiante para diseñar interfaces pequeñas y específicas, en lugar de una interfaz grande que las clases deban implementar en su totalidad.  La habilidad del estudiante para aplicar la inversión de dependencia para reducir el acoplamiento entre módulos de alto y bajo nivel. estudiante para | El estudiante demuestra una comprensión excepcional y una aplicación innovadora de los principios SOLID en el diseño y desarrollo de software. Diseña sistemas que son altamente cohesivos, fácilmente mantenibles y extensibles. Identifica y aplica de manera proactiva oportunidades para mejorar el diseño del software utilizando estos principios, resultando en un código que es robusto frente a cambios y fácil de entender. | El estudiante muestra una sólida comprensión y aplicación efectiva de los principios SOLID. Utiliza estos principios para guiar el diseño de software, lo que resulta en un sistema bien estructurado que apoya la escalabilidad y la flexibilidad. Demuestra habilidad para refacturar el código existente para alinearlo mejor con los principios SOLID, mejorando así la calidad general del software. | El estudiante tiene una comprensión básica de los principios SOLID y puede aplicarlos en situaciones simples. Sin embargo, la aplicación puede ser inconsistente y el diseño del software aún puede presentar problemas de cohesión, acoplamiento, o extensibilidad. Puede haber dificultades para identificar oportunidades para aplicar estos principios en el diseño o refacturación de sistemas existentes. | El estudiante muestra una comprensión limitada y una aplicación inadecuada de los principios SOLID. Hay una falta significativa de comprensión de cómo estos principios contribuyen a un diseño de software de alta calidad. El diseño y desarrollo de software realizado por el estudiante tiende a ser rígido, difícil de mantener y extender, y propenso a errores durante la modificación o extensión del código. |