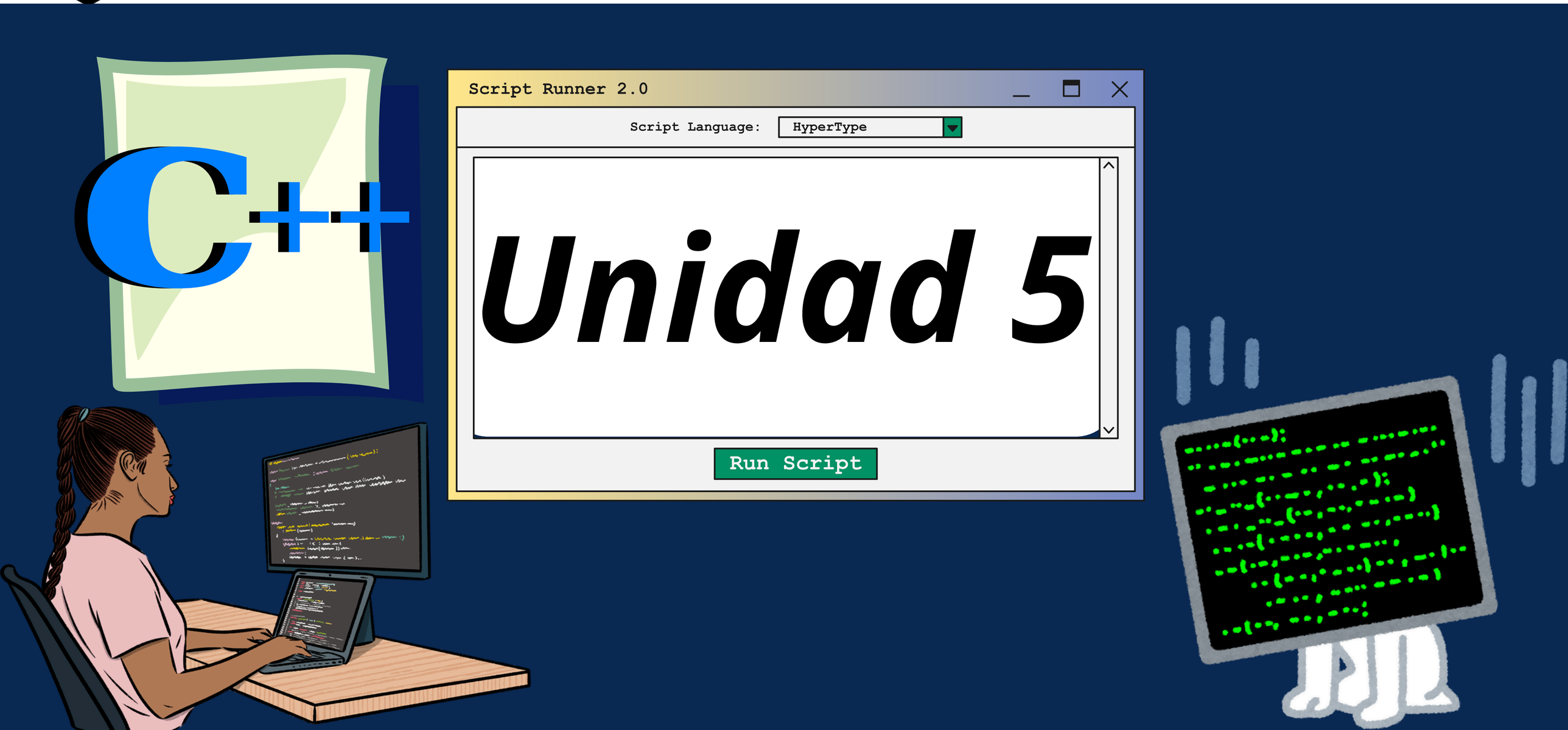
PROGRAMACION AVANZADA

ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz

FACULTAD DE INGENIERIA
UNIVERSIDAD NACIONAL DE JUJUY

PROGRAMACION ORIENTADA A OBJETO



PROGRAMACION ORIENTADA A OBJETO

Ventajas de la Programación Orientada a Objetos (POO)

1. Modularidad: Facilita el mantenimiento al organizar el código en módulos (clases).
2. Reutilización: Permite reutilizar código mediante herencia.
3. Abstracción: Oculta la complejidad interna de los objetos.
4. Encapsulamiento: Protege los datos de accesos no autorizados.
5. Polimorfismo: Flexibiliza el comportamiento de las funciones según el objeto.
6. Facilidad para modelar entidades: Más intuitiva al representar el mundo real.
7. Colaboración: Permite que varios desarrolladores trabajen en paralelo.

PROGRAMACION ORIENTADA A OBJETO

Desventajas de la Programación Orientada a Objetos (POO)

1. Curva de aprendizaje: Más difícil de entender para principiantes.
2. Mayor consumo de memoria: Los objetos pueden ocupar más espacio.
3. Complejidad en proyectos pequeños: La POO puede ser innecesariamente compleja para tareas simples.
4. Rendimiento menor: Puede ser menos eficiente que otros paradigmas en términos de tiempo y espacio.
5. Diseño complejo: En proyectos grandes, las relaciones entre objetos pueden complicarse.
6. Tiempos de compilación más largos: El uso de clases y dependencias puede aumentar el tiempo de compilación.

PROGRAMACION ORIENTADA A OBJETO

Principios de la Programación Orientada a Objetos:

Clases y Objetos:

- **Clase:** Es una plantilla o un modelo que define atributos (datos o propiedades) y métodos (funciones o comportamientos) que los objetos de esa clase tendrán.
- **Objeto:** Es una instancia de una clase. Es la entidad en tiempo de ejecución que se crea a partir de la clase, con valores específicos para sus atributos y que puede realizar acciones mediante sus métodos.

PROGRAMACION ORIENTADA A OBJETO

Clase en Programación Orientada a Objetos (POO)

Una clase es una plantilla o modelo a partir del cual se crean objetos. Define las características y comportamientos comunes que tendrán los objetos que pertenecen a ella.

- Características: Se representan mediante atributos (también llamados propiedades o variables de clase).
- Comportamientos: Se definen a través de métodos (funciones dentro de la clase).

Por ejemplo, si tienes una clase "Coche", sus atributos podrían ser marca, modelo, color, y sus métodos podrían ser arrancar(), frenar(), acelerar().

PROGRAMACION ORIENTADA A OBJETO

Objeto en Programación Orientada a Objetos (POO)

Un objeto es una instancia de una clase. Es decir, es una entidad concreta que se crea basándose en la clase. Mientras que la clase define las características y comportamientos generales, el objeto tiene valores específicos para esos atributos.

Por ejemplo, si la clase es "Coche", un objeto podría ser un coche específico como:

- marca: "Toyota"
- modelo: "Corolla"
- color: "Rojo"

El objeto coche1 es una instancia de la clase Coche, con valores concretos para los atributos de la clase. Además, este objeto puede ejecutar los métodos definidos en la clase, como arrancar() o acelerar().

PROGRAMACION ORIENTADA A OBJETO

En Programación Orientada a Objetos (POO), los objetos son instancias de clases y representan entidades del mundo real o conceptos abstractos que tienen propiedades y comportamientos. Para identificar un objeto correctamente, se consideran tres elementos principales:

PROGRAMACION ORIENTADA A OBJETO

1. Identidad:

Cada objeto tiene una identidad única, que lo distingue de otros objetos, incluso si tienen los mismos atributos o comportamientos. La identidad es algo que no necesariamente se refleja en los atributos del objeto, sino que es una característica inherente de cada instancia.

Por ejemplo, si dos máquinas (objetos de la clase Maquina) tienen el mismo nombre, el mismo número de horas de operación y el mismo consumo de energía, siguen siendo objetos diferentes porque son instancias distintas, con sus propios identificadores en la memoria.

PROGRAMACION ORIENTADA A OBJETO

2. Estado:

El estado de un objeto está compuesto por sus atributos o propiedades (también llamados campos). Estos atributos almacenan la información que describe al objeto. En nuestro ejemplo anterior de una Maquina, los atributos serían:

- Nombre de la máquina.
- Horas de operación.
- Consumo de energía por hora.

El estado de un objeto puede cambiar a lo largo del tiempo (por ejemplo, las horas de operación de una máquina aumentan).

PROGRAMACION ORIENTADA A OBJETO

3. Comportamiento:

El comportamiento de un objeto está determinado por los métodos o funciones definidos en su clase. Estos métodos permiten interactuar con el objeto y modificar su estado. Por ejemplo, un método en la clase Maquina podría calcular el consumo total de energía de la máquina o modificar sus horas de operación.

PROGRAMACION ORIENTADA A OBJETO

```
Maquina maquina1("Torno CNC", 5, 1.5);
Maquina maquina2("Torno CNC", 5, 1.5);
```

Aunque maquina1 y maquina2 tienen los mismos valores en sus atributos (nombre, horas_operacion, consumo_energia), son objetos distintos porque son dos instancias diferentes de la clase Maquina. La identidad de cada objeto es única, lo que les permite existir simultáneamente, incluso si tienen el mismo estado.

PROGRAMACION ORIENTADA A OBJETO

La identificación de objetos en POO es crucial porque asegura que cada objeto sea tratado como una entidad única, incluso si sus propiedades (estado) son iguales a las de otros objetos. Esta característica es fundamental para gestionar correctamente el comportamiento y la interacción de los objetos dentro de un programa orientado a objetos.

PROGRAMACION ORIENTADA A OBJETO

Un objeto es una instancia de una clase y representa una entidad concreta del mundo real o un concepto abstracto. Los objetos agrupan datos (estado) y comportamientos (métodos) que permiten modelar estas entidades en un programa.

Propiedades de los Objetos:

- Estado: El estado del objeto está definido por sus atributos (datos).
- Comportamiento: El comportamiento del objeto está definido por los métodos (funciones) que actúan sobre sus atributos.

PROGRAMACION ORIENTADA A OBJETO

2. Atributos

Los atributos son las propiedades o características que tiene un objeto. Son variables que describen el estado de ese objeto en particular.

- Definición: Un atributo es una variable definida dentro de una clase, que almacena datos relevantes para el objeto. Estos atributos pueden tener diferentes tipos de acceso (público, privado o protegido) según la visibilidad y el encapsulamiento que se quiera aplicar.

PROGRAMACION ORIENTADA A OBJETO

- **Objetos:** Representan una entidad específica y concreta, como un Maquina, un Empleado, o un Producto.
- **Atributos:** Son las características o propiedades de un objeto, como el nombre, las horas de operación o el consumo de energía de una máquina. Los atributos definen el estado del objeto.

Con estos conceptos, podemos estructurar programas orientados a objetos de manera eficiente, agrupando tanto los datos como los comportamientos relacionados en objetos.

PROGRAMACION ORIENTADA A OBJETO

Comportamientos

El comportamiento se refiere a las acciones o funcionalidades que un objeto puede realizar. En términos de POO, estos comportamientos se definen a través de métodos dentro de la clase. Los métodos permiten que los objetos interactúen con sus atributos y entre sí.

Los Comportamientos: Acciones que un objeto puede realizar, definidas a través de métodos en su clase.

PROGRAMACION ORIENTADA A OBJETO

Características de la POO

- ABSTRACCIÓN
- ENCAPSULAMIENTO
- MENSAJES
- POLIMORFISMO
- HERENCIA

PROGRAMACION ORIENTADA A OBJETO

1. Abstracción

La abstracción es el principio que permite representar entidades del mundo real mediante clases, enfocándose en los aspectos relevantes y omitiendo detalles innecesarios. La idea es modelar objetos del mundo real sin necesidad de incluir todas sus características.

- Ejemplo: En una clase Vehículo, podrías incluir atributos como marca, modelo y métodos como mover() y detener(), pero no necesitas representar todos los detalles del funcionamiento interno de un motor.

PROGRAMACION ORIENTADA A OBJETO

2. Encapsulamiento

El encapsulamiento es la técnica que consiste en agrupar datos y métodos que operan sobre esos datos en una única unidad (clase), y controlar el acceso a ellos. Esto se logra definiendo los atributos como privados y proporcionando métodos públicos (getters y setters) para interactuar con esos atributos.

- Beneficios:
  - Mejora la seguridad de los datos, ya que los atributos no pueden ser modificados directamente desde fuera de la clase.
  - Facilita el mantenimiento y la modificación del código, ya que los cambios en la implementación interna no afectan a otras partes del programa.
- Ejemplo: En una clase CuentaBancaria, los atributos como saldo podrían ser privados, y podrías proporcionar métodos públicos como depositar() y retirar() para manipular el saldo.

PROGRAMACION ORIENTADA A OBJETO

3. Mensajes

El concepto de mensajes se refiere a la forma en que los objetos se comunican entre sí. En POO, los objetos envían mensajes a otros objetos para solicitar que ejecuten métodos. Este enfoque promueve la interacción entre objetos, lo que permite que el sistema sea más modular y flexible.

- Ejemplo: Si tienes un objeto Coche y quieres que acelere, envías un mensaje al objeto Coche para que ejecute su método acelerar(). La comunicación entre objetos se basa en el envío de mensajes, permitiendo que cada objeto actúe de manera autónoma.

PROGRAMACION ORIENTADA A OBJETO

Formas de Conocimiento en Programación Orientada a Objetos

En la Programación Orientada a Objetos (POO), un objeto necesita conocer a otros objetos para interactuar con ellos de manera efectiva. Para establecer esta relación, es fundamental que un objeto pueda referirse a otro mediante un nombre, lo que se conoce como ligadura (binding). Esta relación se basa en diferentes formas de conocimiento que permiten a los objetos comunicarse y colaborar. A continuación, se presentan las formas principales de conocimiento en POO:

PROGRAMACION ORIENTADA A OBJETO

1. Conocimiento Interno

El conocimiento interno se refiere a las variables de instancia de un objeto. Estas variables son atributos que pertenecen a la clase y son específicas de cada objeto. A través de estas variables, un objeto puede mantener su propio estado y acceder a datos que son relevantes solo para él. Por ejemplo, en una clase Coche, las variables de instancia pueden incluir el color, el modelo y la velocidad actual. Este conocimiento permite que el objeto tenga una representación interna de sus características y estado.

PROGRAMACION ORIENTADA A OBJETO

2. Conocimiento Externo

El conocimiento externo se refiere a los parámetros que se pasan a los métodos de un objeto. Estos parámetros permiten que un objeto interactúe con otros objetos o modifique su estado basado en información externa. Al recibir parámetros, el objeto puede utilizar datos proporcionados por otros objetos o por el usuario. Por ejemplo, un método acelerar en la clase Coche podría aceptar un parámetro que indique la cantidad de velocidad a incrementar, permitiendo así que el objeto responda a solicitudes externas de manera dinámica.

PROGRAMACION ORIENTADA A OBJETO

3. Pseudo Variables

Las pseudo variables son una forma especial de conocimiento en POO. Estas no son variables en el sentido tradicional, sino que representan valores que son automáticamente entendidos por el sistema o el lenguaje de programación. Un ejemplo común de pseudo variables son los punteros this en C++, que representan la instancia actual del objeto que está ejecutando un método. Esto permite que un objeto acceda a su propia información y se refiera a sí mismo en contextos donde es necesario.

PROGRAMACION ORIENTADA A OBJETO

Variables de Instancia

Las variables de instancia son atributos que pertenecen a un objeto específico dentro de una clase. Definen una relación directa entre un objeto y sus características o propiedades. Estas variables se declaran explícitamente como parte de la estructura de la clase y son únicas para cada instancia del objeto. La relación entre el objeto y sus variables de instancia perdura durante la vida del objeto, es decir, mientras el objeto esté en existencia en la memoria, o hasta que se modifiquen explícitamente por alguna operación o método. Esto permite que cada objeto mantenga su propio estado y comportamiento, diferenciándose de otros objetos de la misma clase.

PROGRAMACION ORIENTADA A OBJETO

Constructores en Programación Orientada a Objetos

Los constructores son funciones miembro especiales en una clase que se utilizan para inicializar un objeto de esa clase en el momento de su creación. A continuación se detallan algunas características clave de los constructores:

- Nombre: Tienen el mismo nombre que la clase a la que pertenecen.
- Tipo de retorno: No tienen tipo de retorno, lo que significa que no retornan ningún valor, ni siquiera void.
- Heredabilidad: No pueden ser heredados, es decir, las clases derivadas no pueden heredar los constructores de sus clases base.
- Visibilidad: Deben ser públicos para que los objetos puedan ser inicializados desde fuera de la clase. No tiene sentido declarar un constructor como privado, ya que se utilizan para crear instancias de la clase.

PROGRAMACION ORIENTADA A OBJETO

Destructores en Programación Orientada a Objetos

Los destructores son funciones miembro especiales en una clase que se utilizan para eliminar un objeto de esa clase cuando ya no es necesario. Su función principal es liberar recursos que pueden haber sido adquiridos durante la vida del objeto, como memoria dinámica o conexiones a bases de datos.

PROGRAMACION ORIENTADA A OBJETO

A continuación se presentan las características clave de los destructores:

- Nombre: Tienen el mismo nombre que la clase, pero precedido por un tilde (~).
- Tipo de retorno: No tienen tipo de retorno y, por lo tanto, no retornan ningún valor.
- Parámetros: No pueden tener parámetros, lo que significa que no se puede pasar información al destructor.
- Heredabilidad: No pueden ser heredados, es decir, las clases derivadas no pueden heredar los destructores de sus clases base.
- Visibilidad: Deben ser públicos, ya que siempre se utilizan desde el exterior de la clase para liberar el objeto.
- Sobrecarga: No pueden ser sobrecargados, dado que no tienen parámetros ni tipo de retorno.

PROGRAMACION ORIENTADA A OBJETO

Seudo Variable: this

La seudo variable this es un concepto fundamental en la programación orientada a objetos que permite a un objeto hacer referencia a sí mismo. Cuando un objeto necesita acceder a sus propios atributos o métodos desde dentro de sus funciones miembro, utiliza esta seudo variable.

PROGRAMACION ORIENTADA A OBJETO

Características de this

- No se declara: this es una variable que no necesita ser declarada; es parte del contexto del objeto.
- No puede modificarse: this siempre apunta al objeto actual y no puede ser cambiado ni reasignado.
- Referencia al objeto actual: Permite que el objeto acceda a sus propios atributos y métodos.

EJEMPLOS

Ejemplo 1: Fundamentos y Encapsulación

Lote de Producción (Clase y Encapsulación)

Concepto Clave: Clase, Objeto y Atributo private.

Consigna: Modele un lote de producción mediante la clase LoteProduccion.

1. Defina la clase LoteProduccion con dos atributos privados: codigoLote (string) y pesoNetoKg (float).
2. Cree un constructor que reciba el código del lote y lo inicialice, y que asigne el peso en 0.0.
3. En la función main(), instancie un objeto llamado lote_A (Objeto).
4. Demuestre mediante un comentario en el código que no es posible modificar el atributo pesoNetoKg directamente desde main(), explicando por qué se aplica la Encapsulación.

EJEMPLOS

Ejemplo 2. Temperatura de Proceso (Setter con Validación de Seguridad)

Concepto Clave: Setter (Método de Establecimiento) y Validación.

Consigna: Implemente un sistema de control de temperatura de un horno industrial.

1. Defina la clase ProcesoTermico con el atributo privado temperaturaActual (float).
2. Cree el método Setter setTemperatura(float temp) (público).
3. Dentro del Setter, implemente la siguiente regla de seguridad industrial: si la temperatura recibida es mayor a 350.0 °C, el método debe mostrar una "ALERTA: Límite excedido" y establecer la temperaturaActual al máximo seguro de 350.0 °C.
4. En main(), pruebe el Setter asignando una temperatura de 450.0 °C y luego muestre el valor final guardado.

EJEMPLOS

Ejemplo 3. Horas de Máquina (Getter Lógico para Mantenimiento)

Concepto Clave: Getter (Método de Obtención) Lógico.

Consigna: Modele una fresadora CNC y su ciclo de mantenimiento.

1. Defina la clase MaquinaCNC con los atributos privados horasAcumuladas (int) y la constante HORAS_MANTENIMIENTO (fije su valor en 500).
2. Cree el método Getter getHorasFaltantesMantenimiento() (público).
3. Este Getter no debe devolver un atributo, sino un cálculo: la diferencia entre HORAS_MANTENIMIENTO y horasAcumuladas. Si el resultado es negativo (ya pasó el límite), debe devolver 0.
4. En main(), instancie una máquina con 350 horas y use el Getter para indicar cuántas horas faltan para la parada programada.

EJEMPLOS

Ejemplo 4. Producto en Inventario (Constructor con Inicialización Segura)

Concepto Clave: Constructor.

Consigna: Asegure que todos los productos de almacén nazcan con un stock mínimo válido.

1. Defina la clase ProductoInventario con los atributos privados codigoSKU (string) y stockMinimo (int).
2. Cree el Constructor ProductoInventario(string sku, int minimo).
3. Dentro del Constructor, implemente una validación inicial: si el parámetro minimo es menor a 0, el Constructor debe mostrar un mensaje de error y forzar el atributo stockMinimo a ser 1. De lo contrario, asigna el valor recibido.
4. En main(), cree un objeto valvula con stock mínimo de 50 y otro objeto fusible con stock mínimo de -10, y muestre el valor final asignado en cada caso.

EJEMPLOS

Ejemplo 5. Alerta de Reposición (Método de Lógica Empresarial)

Concepto Clave: Encapsulación de Lógica.

Consigna: Modele la lógica de decisión para la reposición de inventario.

1. Defina la clase ProductoAlmacen con atributos privados stockActual (int) y stockSeguridad (int).
2. Cree un método público requiereReposicion() que no recibe parámetros y devuelve un valor bool.
3. Implemente la regla: la función debe devolver true si el stockActual es menor o igual al stockSeguridad.
4. En main(), cree un objeto, asigne valores (ej: Stock Actual=80, Stock Seguridad=100) y use el método requiereReposicion() dentro de una sentencia if para imprimir un mensaje de "¡Alerta de Compra!".

EJEMPLOS

Ejemplo 6. Costo de Materia Prima (this para Evitar Ambigüedad)

Concepto Clave: Seudovariable this.

Consigna: Modifique el costo unitario de una materia prima usando un Setter que tenga el mismo nombre del atributo.

1. Defina la clase MateriaPrima con el atributo privado float costo;.
2. Cree el Setter público void setCosto(float costo) donde el parámetro de entrada se llama igual que el atributo.
3. Dentro del Setter, use la seudovariable this para asignar correctamente el valor: this->costo = costo;.
4. En main(), instancie un objeto y llame al setCosto() dos veces con valores diferentes, demostrando que this permitió la asignación sin problemas de ambigüedad.

EJEMPLOS

Ejemplo 7. Asignación de Turno (Constructor con Decisión)

Concepto Clave: Lógica Condicional en Constructor.

Consigna: Asigne el turno de un operario automáticamente al ser creado, basándose en su hora de entrada.

1. Defina la clase Operario con los atributos privados nombre (string) y turnoAsignado (string).
2. Cree un Constructor que reciba el nombre y la horaEntrada (int).
3. Dentro del Constructor, use estructuras de decisión (if/else if) para asignar el turnoAsignado con la siguiente lógica:
   - Si horaEntrada está entre 6 y 13 (inclusive): "Mañana".
   - Si horaEntrada está entre 14 y 21 (inclusive): "Tarde".
   - Cualquier otra hora (simulando noche/madrugada): "Noche".
4. En main(), cree dos objetos: operario_A (con hora 7) y operario_B (con hora 15), y muestre su turno asignado.

EJEMPLOS

Ejemplo 8. Rendimiento (Setter con Aco-tamiento de Valores)

Concepto Clave: Setter con Validación de Rango (Acotamiento).

Consigna: Controle el porcentaje de rendimiento de un proceso para que siempre se mantenga entre 0% y 100%.

1. Defina la clase IndicadorKPI con el atributo privado rendimientoPorcentual (float).
2. Cree el Setter público setRendimiento(float rendimiento).
3. Dentro del Setter, aplique la siguiente validación de acotamiento:
    - Si el valor es menor a 0, fíjelo en 0.0.
    - Si el valor es mayor a 100, fíjelo en 100.0.
    - Solo si está entre 0 y 100, asigne el valor recibido.
4. En main(), pruebe el Setter intentando asignar valores de 95.5, luego 110.0 y finalmente -5.0. Muestre el valor final guardado en cada caso para demostrar que el Setter lo corrigió.

¿PREGUNTAS?

MUCHAS

GRACIAS!!!

Nos Vemos en la Siguiente Clase