

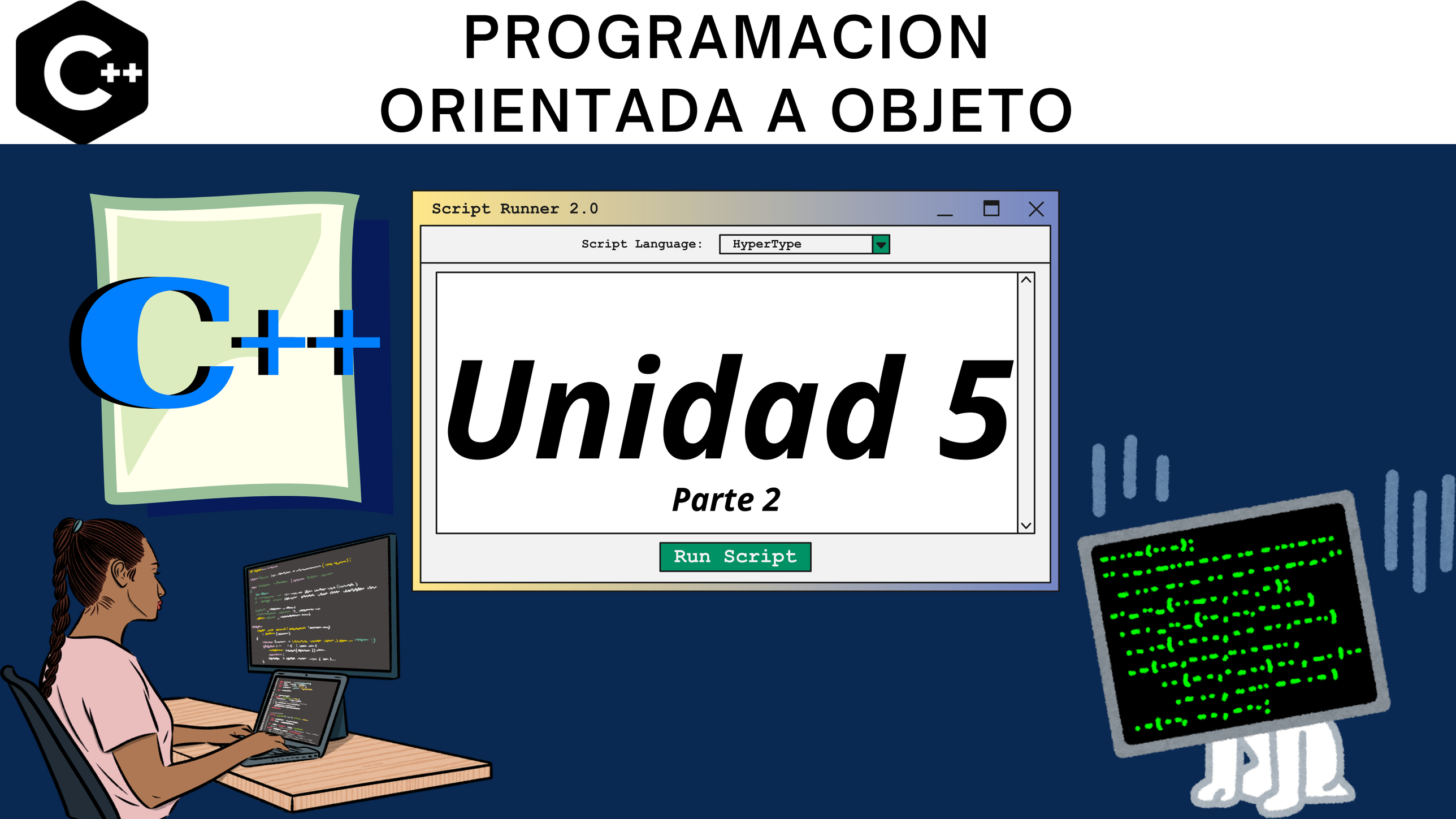
# PROGRAMACION AVANZADA

ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz

FACULTAD DE INGENIERIA
UNIVERSIDAD NACIONAL DE JUJUY

# PROGRAMACION ORIENTADA A OBJETO





# PROGRAMACION ORIENTADA A OBJETOS

En programación orientada a objetos, los miembros de una clase (atributos y métodos) pueden tener distintos niveles de visibilidad, lo que determina si pueden ser accedidos desde fuera de la clase. La visibilidad se establece mediante tres secciones: pública, privada y protegida.



# PROGRAMACION ORIENTADA A OBJETOS

Sección Privada (private): Esta sección contiene los miembros de la clase que solo pueden ser accedidos desde dentro de la misma clase. No se puede acceder a estos atributos o métodos desde el exterior, lo que ayuda a evitar errores accidentales y protege los datos. Se suele utilizar private para los datos de la clase, asegurando que solo se modifiquen a través de métodos específicos.



# PROGRAMACION ORIENTADA A OBJETOS

Sección Pública (public): En esta sección se encuentran los miembros de la clase que pueden ser accedidos desde cualquier parte del programa. Los métodos y atributos definidos como públicos pueden ser utilizados tanto dentro como fuera de la clase. Usualmente, aquí se declaran los métodos que permiten interactuar con los datos privados de la clase.



# PROGRAMACION ORIENTADA A OBJETOS

Sección Protegida (protected): Los miembros definidos como protected no son accesibles directamente desde el exterior, pero están disponibles para las clases derivadas (clases que heredan de la clase base). Esto permite que las subclases tengan acceso directo a ciertos datos o métodos sin exponerlos completamente al exterior.



# PROGRAMACION ORIENTADA A OBJETOS

En la práctica, los atributos de una clase suelen estar en la sección privada para protegerlos, mientras que los métodos que los manejan suelen estar en la sección pública, para permitir el acceso controlado a los datos.



# PROGRAMACION ORIENTADA A OBJETOS

En C++, la visibilidad de los miembros de una clase (atributos y métodos) se define mediante las palabras clave **public**, **private** y **protected**, que determinan el nivel de acceso que otros componentes del programa pueden tener a estos miembros. La estructura general para definir la visibilidad en una clase es la siguiente:



# PROGRAMACION ORIENTADA A OBJETOS

```
class <NombreClase> {
public:
        // Atributos y métodos públicos
        // Estos miembros pueden ser accedidos desde cualquier parte del programa.
protected:
        // Atributos y métodos protegidos
        // Estos miembros son accesibles solo desde la clase y sus subclases, pero no desde el exterior.
private:
        // Atributos y métodos privados
        // Estos miembros solo pueden ser accedidos desde dentro de la misma clase, no desde el exterior. };
```



# PROGRAMACION ORIENTADA A OBJETOS

- **public:** Los miembros definidos en esta sección son accesibles desde cualquier parte del programa. Esto se utiliza para métodos y atributos que deben estar disponibles públicamente.
- **protected:** Los miembros en esta sección son accesibles solo desde la clase misma y las clases derivadas. No son visibles desde el exterior, pero las subclases pueden acceder a ellos.
- **private:** Los miembros aquí declarados solo pueden ser accedidos desde dentro de la misma clase. Son invisibles para el resto del programa y protegen los datos de modificaciones accidentales.

La organización de estos tres niveles de visibilidad dentro de una clase permite controlar el acceso y garantizar la integridad de los datos, proporcionando una capa de encapsulamiento en el diseño de clases en C++.



# PROGRAMACION ORIENTADA A OBJETOS

En la definición de una clase, se pueden utilizar las secciones public, protected y private para establecer el nivel de acceso a sus miembros, aunque no es obligatorio incluirlas ni seguir un orden específico.

Existen tres tipos de usuarios para una clase:

- La propia clase.
- Usuarios externos o genéricos.
- Clases derivadas.

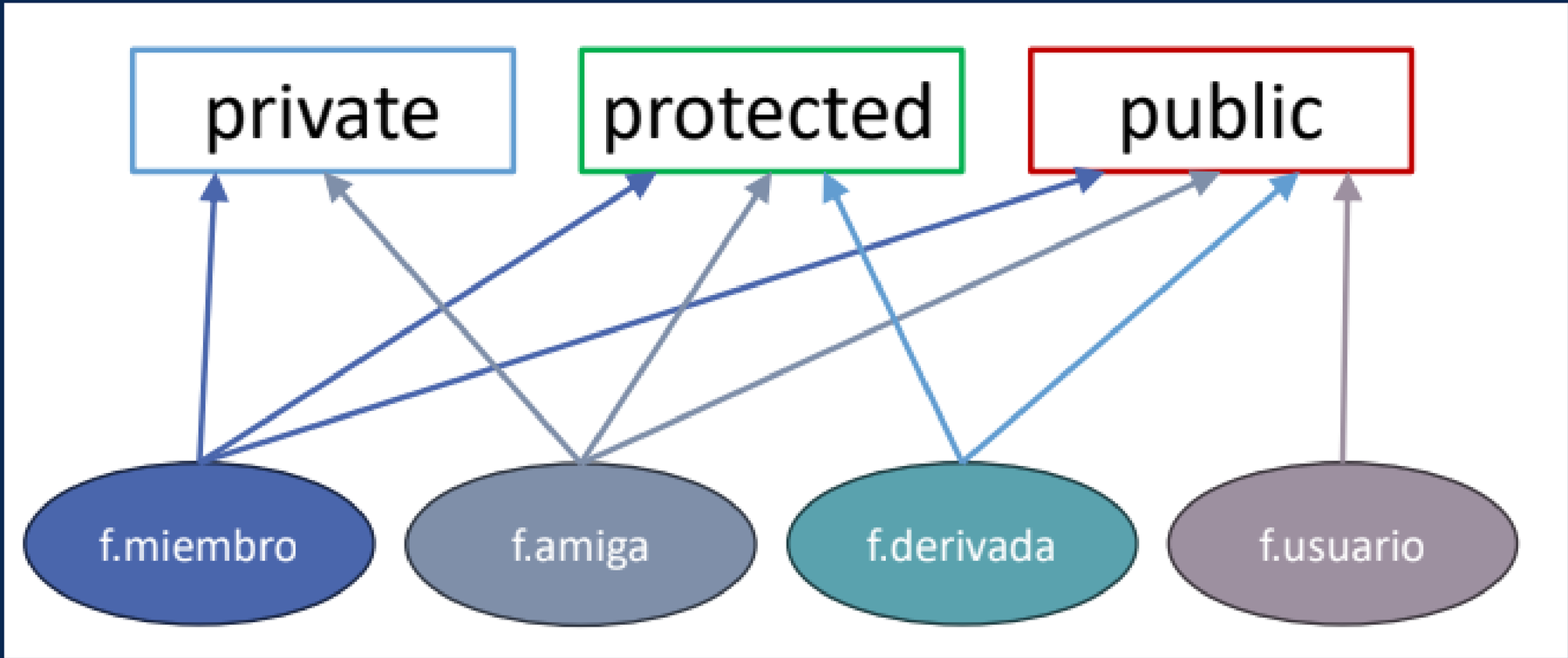
Cada tipo de visibilidad se define con una palabra clave:

- private: Solo la propia clase puede acceder a estos miembros.
- public: Permite acceso a cualquier usuario.
- protected: Accesible únicamente desde clases derivadas y la propia clase.

Los métodos dentro de la clase tienen acceso completo a todos sus miembros. Además, algunas funciones externas, llamadas "amigas" (friend), también pueden acceder a los miembros privados y protegidos de la clase.

# PROGRAMACION ORIENTADA A OBJETOS

**Modos de Acceso**





# PROGRAMACION ORIENTADA A OBJETOS

## *Encapsulamiento*

- Una clase debe mantener ocultos sus detalles internos y la implementación de sus funcionalidades, exponiendo solo las interfaces de sus servicios de manera pública.
- Los atributos de la clase deben ser privados, permitiendo que el acceso desde otras partes del programa se realice únicamente mediante métodos públicos.
- Los métodos que se utilicen para manipular datos internamente o que sirvan como soporte a otros métodos de la misma clase deben definirse como privados.
- Los atributos de un objeto deben estar protegidos y su acceso debe realizarse exclusivamente a través de métodos públicos.



# PROGRAMACION ORIENTADA A OBJETOS

## Constructores

Los constructores se utilizan para inicializar un objeto en el momento de su creación. Algunas de sus características principales son las siguientes:

- Tienen el mismo nombre que la clase.
- Pueden definirse dentro de la declaración de la clase (inline) o fuera de ella.
- Pueden recibir parámetros, los cuales también pueden tener valores por defecto.
- Es posible sobrecargar constructores. Esto significa que se pueden definir múltiples constructores en la misma clase, siempre que tengan listas de argumentos diferentes. Esto permite inicializar los objetos de diferentes maneras.
- No tienen un valor de retorno. No es posible especificar un tipo de retorno, ni siquiera void, y no pueden contener ninguna instrucción return.
- Se ejecutan automáticamente cuando se crea un objeto de la clase, sin necesidad de llamarlos explícitamente.
- Generalmente, se declaran en la sección pública para que el constructor pueda ser utilizado por cualquier código que cree un objeto de la clase. Si estuvieran en la sección privada, solo podrían ser usados dentro de las funciones miembro o funciones amigas de la clase.

El constructor que no recibe ningún argumento se conoce como constructor por defecto.



# PROGRAMACION ORIENTADA A OBJETOS

## Constructores de Copia

Un constructor de copia es un tipo especial de constructor que permite crear un nuevo objeto utilizando los datos de un objeto ya existente. Este constructor recibe como argumento una referencia constante a un objeto de la misma clase.

Los constructores de copia se invocan de manera implícita en las siguientes situaciones:

- Cuando se realizan asignaciones entre objetos.
- Cuando se pasan objetos como parámetros por valor a funciones.

La relevancia de los constructores de copia radica en que:

- Son el único método disponible para duplicar un objeto.
- Se utilizan cuando los objetos se pasan como parámetros a funciones.
- Son esenciales cuando una función necesita devolver un objeto.

# PROGRAMACION ORIENTADA A OBJETOS

## Funciones Friend en C++

Las funciones que no son miembros de una clase pueden acceder a los atributos privados de esa clase si se declaran como funciones amigas (friend). La declaración se realiza dentro de la clase utilizando la palabra clave friend, mientras que la implementación de la función puede estar en cualquier lugar del código. Cabe destacar que las funciones amigas no pueden ser definidas como funciones inline.

La sintaxis para declarar funciones amigas es la siguiente:

```
class Ejemplo {
    // Atributos y métodos de la clase
public:
    friend tipo funcion(parametros); // Prototipo de la función amiga
};
```



# PROGRAMACION ORIENTADA A OBJETOS

## Herencia

La herencia es una característica fundamental de la programación orientada a objetos que permite derivar nuevas clases de clases ya existentes. A través de la herencia, se pueden reutilizar las propiedades y métodos de la clase base (o padre) en la nueva clase (o hija), a la vez que se pueden agregar nuevas funcionalidades o modificar las existentes. Esto promueve la reutilización de código y la organización jerárquica de las clases.



# PROGRAMACION ORIENTADA A OBJETOS

## Herencia

La herencia es una característica fundamental de la programación orientada a objetos que permite derivar nuevas clases de clases ya existentes. A través de la herencia, se pueden reutilizar las propiedades y métodos de la clase base (o padre) en la nueva clase (o hija), a la vez que se pueden agregar nuevas funcionalidades o modificar las existentes. Esto promueve la reutilización de código y la organización jerárquica de las clases.



# PROGRAMACION ORIENTADA A OBJETOS

## Jerarquía de Clases: Clases Base y Clases Derivadas

En el contexto de la herencia, una clase derivada es aquella que se crea a partir de una o varias clases existentes, conocidas como clases base. Este sistema jerárquico permite que las clases derivadas hereden atributos y métodos de sus clases base, y también pueden servir como base para nuevas clases derivadas. La herencia múltiple se refiere a la capacidad de una clase derivada de heredar de más de una clase base. Este enfoque facilita la creación de jerarquías de clases complejas, permitiendo encapsular diferentes aspectos de un objeto, al tiempo que se establecen conexiones entre objetos más especializados que comparten un tipo básico y heredan sus características.

# PROGRAMACION ORIENTADA A OBJETOS

Derivar clases, sintaxis La forma general de declarar clases derivadas es la siguiente:

```
class ClaseDerivada : public ClaseBase {
    // Miembros de la clase derivada
};
```

Descripción de la Sintaxis:

- class ClaseDerivada: Define una nueva clase llamada ClaseDerivada.
- : public ClaseBase: Indica que ClaseDerivada hereda de ClaseBase utilizando herencia pública. Esto significa que los miembros públicos y protegidos de ClaseBase se convierten en miembros públicos y protegidos de ClaseDerivada, respectivamente.
- { ... }: Dentro de las llaves se pueden definir los miembros (atributos y métodos) específicos de la ClaseDerivada.



# PROGRAMACION ORIENTADA A OBJETOS

## Constructores de Clases Derivadas

Al instanciar un objeto de una clase derivada, el primer paso es llamar al constructor de la clase base correspondiente. Una vez que el constructor de la clase base se ha ejecutado, se procede a invocar el constructor de la clase derivada. Si la clase base también hereda de otra clase, este proceso se repetirá en forma recursiva hasta alcanzar la raíz de la jerarquía de clases. En caso de que no se hayan definido constructores específicos, se utilizarán automáticamente los constructores por defecto generados por el compilador.

# EJEMPLO 1: HERENCIA SIMPLE (ESTACIONES DE TRABAJO)

Herencia pública y atributos heredados.

Modele las estaciones de una línea de montaje industrial.

1. Defina la clase base EstacionTrabajo con atributos numeroEstacion (int) y responsable (string).
2. Defina la clase derivada EstacionRobotizada que herede públicamente de EstacionTrabajo y añada el atributo modeloRobot (string).
3. Cree un Constructor en EstacionRobotizada que inicialice todos los atributos (propios y heredados).
4. En main(), cree un objeto de EstacionRobotizada y muestre sus datos utilizando los métodos de la clase base.

# EJEMPLO 2: VISIBILIDAD PROTECTED (CONTROL DE TIEMPOS)

Diferencia de acceso entre private y protected.

Analice los tiempos de ciclo en un proceso de fabricación.

1. Defina la clase base ProcesoFabricacion con un atributo privado costoHoraHombre (float) y un atributo protegido codigoLinea (string).
2. Defina la clase derivada ProcesoMecanizado.
3. Cree un método en la clase derivada que intente acceder tanto a codigoLinea como a costoHoraHombre.
4. En main(), demuestre mediante un comentario de código que los atributos protected no se pueden modificar directamente desde fuera de la clase.

# EJEMPLO 3: CONSTRUCTOR DE CLASE DERIVADA (SUMINISTROS QUÍMICOS)

Lista de Inicialización en constructores con herencia.

Modele el ingreso de suministros químicos para tratamientos térmicos.

1. Defina la clase base Suministro con un constructor que inicialice codigoLote (string) y volumenLitros (float).
2. Defina la clase derivada SuministroQuimico que añada temperaturaMax (float).
3. El Constructor de SuministroQuimico debe recibir la temperatura máxima más los parámetros de la base, utilizando obligatoriamente la Lista de Inicialización.
4. En main(), cree un objeto y muestre todos sus atributos.

# EJEMPLO 4: REDEFINICIÓN DE MÉTODOS (MANTENIMIENTO DE MAQUINARIA)

Especialización de comportamiento en subclases.

Modele el plan de mantenimiento preventivo para distintos tipos de máquinas.

1. Defina la clase base EquipoPlanta con un identificador de activo.
2. Defina dos clases derivadas: PrensaHidraulica y CintaTransportadora.
3. Implemente el método público programarMantenimiento() en ambas clases derivadas con una lógica de impresión especializada y adaptada a cada máquina.
4. En main(), cree objetos de ambas clases y ejecute sus métodos.



# EJEMPLO 5: HERENCIA Y VALIDACIÓN (LÍNEA DE EMBALAJE)

Uso de atributos heredados en reglas de negocio.

Controle el límite de peso en cajas destinadas al sector de expedición.

1. Defina la clase base EmbalajeEstandar con los atributos pesoMaximoPermitidoKg y tipoCaja.
2. Defina la clase derivada CajaExportacion que añada el atributo pesoActualKg.
3. Cree un método Setter pesarYVerificar(float peso) en la derivada que use pesoMaximoPermitidoKg (heredado) para validar si se excede la capacidad de carga de la caja. Si se pasa, muestra alerta y ajusta al límite.
4. En main(), pruebe el comportamiento con cargas normales y excesivas.

# EJEMPLO 6: JERARQUÍA DE 3 NIVELES (EQUIPOS DE PROTECCIÓN - EPP)

Herencia multinivel (Base -> Derivada -> Derivada).

Diseñe una jerarquía de clases para la gestión de Equipos de Protección Personal (EPP).

1. Nivel 1: Clase base ElementoProteccion con un atributo protegido normativaISO.
2. Nivel 2: Clase ProteccionFacial que hereda de ElementoProteccion y añade materialVisor.
3. Nivel 3: Clase CaretaSoldar que hereda de ProteccionFacial y añade gradoOscurecimiento.
4. El constructor del Nivel 3 debe inicializar toda la cadena de herencia de forma correcta.

# EJEMPLO 7: DESTRUCTOR EN HERENCIA (CIERRE DE LÍNEA DE PRODUCCIÓN)

Orden de ejecución de destructores (de objeto derivado a base). Simule el apagado secuencial y seguro de una celda robotizada de manufactura.

1. Defina la clase base CeldaBase y la clase derivada CeldaAutomatizada.
2. Implemente destructores en ambas clases (~CeldaBase() y ~CeldaAutomatizada()) que impriman mensajes indicando el cierre de recursos.
3. En main(), cree una instancia local de la clase derivada dentro de un bloque de ámbito.
4. Observe el orden de los mensajes en pantalla para comprobar que el destructor de la clase derivada se ejecuta primero.

# EJEMPLO 7: DESTRUCTOR EN HERENCIA (CIERRE DE LÍNEA DE PRODUCCIÓN)

Orden de ejecución de destructores (de objeto derivado a base). Simule el apagado secuencial y seguro de una celda robotizada de manufactura.

1. Defina la clase base CeldaBase y la clase derivada CeldaAutomatizada.
2. Implemente destructores en ambas clases (~CeldaBase() y ~CeldaAutomatizada()) que impriman mensajes indicando el cierre de recursos.
3. En main(), cree una instancia local de la clase derivada dentro de un bloque de ámbito.
4. Observe el orden de los mensajes en pantalla para comprobar que el destructor de la clase derivada se ejecuta primero.

# EJEMPLO 8: VISIBILIDAD PROTECTED PARA GETTERS (CONTROL DE PRODUCTIVIDAD)

Métodos protegidos accesibles dentro de la jerarquía de clases.

Calcule la eficiencia de mano de obra mediante un método protegido en la base.

1. Defina la clase base RegistroPersonal con el atributo privado horasEfectivas (float).
2. Declare un método Getter getHorasEfectivas() en la sección protected de la clase base.
3. Defina la clase derivada ReporteEficienciaOperario que herede públicamente de RegistroPersonal.
4. Cree el método calcularRendimiento(float horasTeoricas) en la clase derivada que invoque al Getter protegido de la base para realizar el cálculo.
5. Demuestre que main() no puede invocar directamente a getHorasEfectivas() por estar protegido.




# PROGRAMACION ORIENTADA A OBJETOS

# FIN...






¿PREGUNTAS?

MUCHAS

# GRACIAS!!!

*Nos Vemos en la Siguiente Clase*