

# **Object Oriented Programming OOP IEC61131-3 Youtube Course by Runtimevic**

---

**Object Oriented Programming OOP IEC61131-3 PLC Youtube Course by  
Runtimevic.**

*runtimevic*

*Copyright © 2023 Víctor Durán.*

## Table of contents

---

1. Requisiti	4
2. Introduzione	5
3. Tipi di paradigmi	6
4. Classi e oggetti	7
4.1 Classi e oggetti	7
4.2 Blocco funzione	8
4.3 Metodo oggetto	12
4.4 Proprietà oggetto	16
4.5 Ereditarietà	17
4.6 THIS puntatore	20
4.7 SUPER puntatore	21
4.8 Interfaccia	22
4.9 puntatore e riferimento	23
4.10 Parola chiave Abstracto	24
4.11 FB Abstract vs. Interfaccia	25
4.12 Interfaccia fluente	26
5. Tabella dei modificatori di accesso	27
6. Tipi di variabili e variabili speciali	28
7. Tipi di dati	31
8. Principi OOP	32
8.1 4 Pilastri	32
8.2 Astrazione	33
8.3 Incapsulamento	34
8.4 Ereditarietà	35
8.5 Polimorfismo	36
9. Relazioni	37
10. SOLID	38
10.1 SOLID	38
10.2 Principio di responsabilità esclusiva	39
10.3 Principio aperto/chiuso	40
10.4 Principio di sostituzione di Liskov	41
10.5 Principio di segregazione dell'interfaccia	42
10.6 Principio di inversione delle dipendenze	43
11. UML	44
11.1 UML	44

11.2	Classe UML	45
11.3	StateChart UML	46
12.	Tipi di progettazione per la programmazione PLC	47
13.	Modelli di progettazione	48
13.1	Modelli di progettazione	48
13.2	Modello di strategia	49
13.3	Modello astratto della fabbrica	50
14.	Links	51

## 1. Requisiti

---



## 2. Introduzione

---



## 3. Tipi di paradigmi

---

help

help

help

## 4. Classi e oggetti

---

### 4.1 Classi e oggetti

---

help

help

help

## 4.2 Blocco funzione

---

### 4.2.1 Blocco funzione

---

help

help

help



#### 4.2.2 Modificatori di accesso ai blocchi funzione

---

help

help

help

## 4.2.3 Function Block Dichiarazione variabili

---

help

help

help

#### 4.2.4 Costruttore e distruttore

---

help

help

help

## 4.3 Metodo oggetto

---

### 4.3.1 Metodo

---

help

help

help

### 4.3.2 Modificatori di accesso ai metodi

---

help

help

help

### 4.3.3 Metodo Dichiarazione delle variabili

---

help

help

help

#### 4.3.4 Tipi di variabili restituite dal metodo

---

help

help

help

## 4.4 Proprietà oggetto

---

help

help

help



## 4.5 Ereditarietà

---

### 4.5.1 Blocco funzione ereditarietà

---

help

help

help

## 4.5.2 Struttura di ereditarietà

---

help

help

help

### 4.5.3 Interfaccia di ereditarietà

---

help

help

help

## 4.6 THIS puntatore

---

help

help

help

**THIS pointer** The THIS pointer is available to all function blocks and points to the current function block instance. This pointer is required whenever a method contains a local variable which obscures a variable in the function block.

An assignment statement within the method sets the value of the local variable. If we want the method to set the value of the local variable in the function block, we need to use the THIS pointer to access it.

## 4.7 SUPER puntatore

---

help

help

help

## 4.8 Interfaccia

---

help

help

help

## 4.9 puntatore e riferimento

---

help

help

help

## 4.10 Parola chiave Abstracto

---



## 4.11 FB Abstract vs. Interfaccia

---

help

help

help

## 4.12 Interfaccia fluente

---

help

help

help

## 5. Tabella dei modificatori di accesso

---

help

help

help

## 6. Tipi di variabili e variabili speciali

---

### 6.0.1 Variable types and special variables:

The variable type defines how and where you can use the variable. The variable type is defined during the variable declaration.

### 6.0.2 Further Information:

- [Local Variables - VAR](#)
- [Input Variables - VAR\\_INPUT](#)
- [Output Variables - VAR\\_OUTPUT](#)
- [Input/Output Variables - VAR\\_IN\\_OUT, VAR\\_IN\\_OUT CONSTANT](#)
- [Global Variables - VAR\\_GLOBAL](#)
- Solo es posible su declaración en GVL (Lista de Variables Global)
- [Temporary Variable - VAR\\_TEMP](#)
- Esta funcionalidad es una extensión con respecto a la norma IEC 61131-3.
- Las variables temporales se declaran localmente entre las palabras clave VAR\_TEMP y END\_VAR.
- VAR\_TEMP declaraciones sólo son posibles en **programas y bloques de funciones**.
- TwinCAT reinicializa las variables temporales cada vez que se llama al bloque de funciones.

La aplicación sólo puede acceder a variables temporales en la parte de implementación de un programa o bloque de funciones. - [Static Variables - VAR\\_STAT](#) - Esta funcionalidad es una extensión con respecto a la norma IEC 61131-3. - Las variables estáticas se declaran localmente entre las palabras clave VAR\_STAT y END\_VAR. TwinCAT inicializa las variables estáticas cuando se llama por primera vez al bloque de funciones respectivo. - Puede tener acceso a las variables estáticas sólo dentro del espacio de nombres donde se declaran las variables (como es el caso de las variables estáticas en C). Sin embargo, las variables estáticas conservan su valor cuando la aplicación sale del bloque de funciones. Puede utilizar variables estáticas, como contadores para llamadas a funciones, por ejemplo. - Puede extender variables estáticas con una palabra clave de atributo. - Las variables estáticas solo existen una vez. Esto también se aplica a las variables estáticas de un bloque de funciones o un método de bloque de funciones, incluso si el bloque de funciones se instancia varias veces. - [External Variables - VAR\\_EXTERNAL](#) - Las variables externas son variables globales que se "importan" en un bloque de funciones. - Puede declarar las variables entre las palabras clave VAR\_EXTERNAL y END\_VAR. Si la variable global no existe, se emite un mensaje de error.

- En TwinCAT 3 PLC no es necesario que las variables se declaren como externas. La palabra clave existe para mantener la compatibilidad con IEC 61131-3. - Si, no obstante, utiliza variables externas, asegúrese de abordar las variables asignadas (con AT %I o AT %Q) sólo en la lista global de variables. El direccionamiento adicional de las instancias de variables locales daría lugar a duplicaciones en la imagen del proceso. - Estas variables declaradas también tiene que estar declarada la misma variable con el mismo nombre en una GVL (Lista de Variables Global) - [Instance Variables - VAR\\_INST](#) - TwinCAT crea una variable VAR\_INST de un método no en la pila de métodos como las variables VAR, sino en la pila de la instancia del bloque de funciones. Esto significa que la variable VAR\_INST se comporta como otras variables de la instancia del bloque de función y no se reinicializa cada vez que se llama al método. - VAR\_INST variables solo están permitidas en los métodos de un bloque de funciones, y el acceso a dicha variable solo está disponible dentro del método. Puede supervisar los valores de las variables de instancia en la parte de declaración del método. - Las variables de instancia no se pueden extender con una palabra clave de atributo. - [Remanent Variables - PERSISTENT, RETAIN](#) Las variables remanentes pueden conservar sus valores más allá del tiempo de ejecución habitual del programa. Las variables remanentes se pueden declarar como variables RETAIN o incluso más estrictamente como variables PERSISTENTES en el proyecto PLC.

Un requisito previo para la funcionalidad completa de las variables RETAIN es un área de memoria correspondiente en el controlador (NovRam). Las variables persistentes se escriben solo cuando TwinCAT se apaga. Esto requerirá generalmente un UPS correspondiente. Excepción: Las variables persistentes también se pueden escribir con el bloque de función FB\_WritePersistentData.

Si el área de memoria correspondiente no existe, los valores de las variables RETAIN y PERSISTENT se pierden durante un corte de energía.

Variables remanentes - PERSISTENT, RETAIN 1:

La declaración AT no debe utilizarse en combinación con VAR RETAIN o VAR PERSISTENT.

Variables persistentes Puede declarar variables persistentes agregando la palabra clave PERSISTENT después de la palabra clave para el tipo de variable (VAR, VAR\_GLOBAL, etc.) en la parte de declaración de los objetos de programación.

Las variables PERSISTENTES conservan su valor después de una terminación no controlada, un Reset cold o una nueva descarga del proyecto PLC. Cuando el programa se reinicia, el sistema continúa funcionando con los valores almacenados. En este caso, TwinCAT reinicializa las variables "normales" con sus valores iniciales especificados explícitamente o con las inicializaciones predeterminadas. En otras palabras, TwinCAT solo reinicializa las variables PERSISTENTES durante un origen de Restablecer.

Un ejemplo de aplicación para variables persistentes es un contador de horas de funcionamiento, que debe continuar contando después de un corte de energía y cuando el proyecto PLC se descarga nuevamente.

Tabla de información general que muestra el comportamiento de las variables PERSISTENTES Después del comando en línea

VAR PERSISTENTE

Restablecer frío

Los valores se conservan

Restablecer origen

Los valores se reinician

Descargar

Los valores se conservan

Cambio en línea

Los valores se conservan

Evite usar el tipo de datos POINTER TO en listas de variables persistentes, ya que los valores de dirección pueden cambiar cuando el proyecto PLC se descargue nuevamente. TwinCAT emite las advertencias correspondientes del compilador. Declarar una variable local como PERSISTENTE en una función no tiene ningún efecto. La persistencia de datos no se puede utilizar de esta manera. El comportamiento durante un restablecimiento en frío puede verse influenciado por el pragma 'TcInitOnReset'

Variables RETAIN Puede declarar variables RETAIN agregando la palabra clave RETAIN después de la palabra clave para el tipo de variable (VAR, VAR\_GLOBAL, etc.) en la parte de declaración de los objetos de programación.

Las variables declaradas como RETAIN dependen del sistema de destino, pero normalmente se administran en un área de memoria separada que debe protegerse contra fallas de energía. El llamado controlador Retain asegura que las variables RETAIN se escriban al final de un ciclo PLC y solo en el área correspondiente de la NovRam. El manejo del controlador de retención se describe en el capítulo "Conservar datos" de la documentación de C/C++.

Las variables RETAIN conservan su valor después de una terminación incontrolada (corte de energía). Cuando el programa se reinicia, el sistema continúa funcionando con los valores almacenados. En este caso, TwinCAT reinicializa las variables "normales" con sus valores iniciales especificados explícitamente o con las inicializaciones predeterminadas. TwinCAT reinicializa las variables RETAIN en un origen de restablecimiento.

Una posible aplicación es un contador de piezas en una planta de producción, que debe seguir contando después de un corte de energía.

Tabla general que muestra el comportamiento de las variables RETAIN Después del comando en línea

RETENCIÓN DE VAR

Restablecer frío

Los valores se conservan

Restablecer origen

Los valores se reinician

Descargar

Los valores se conservan - [SUPER](#) - [THIS](#) - [Variable types - attribute keywords](#) - [RETAIN](#); for remanent variables of type [RETAIN](#) - [PERSISTENT](#); for remanent variables of type [PERSISTENT](#) - [CONSTANT](#); for constants

---

## 6.0.3

- [infosys.beckhoff.com/](#)
- [infosys.beckhoff.com/](#)
- [www.plccoder.com/instance-variables-with-var\\_inst](#)
- [www.plccoder.com/var\\_temp-var\\_stat-and-var\\_const](#)
- [Tipos de variables y variables especiales](#) help

help

help

## 7. Tipi di dati

---

help

help

help

## 8. Principi OOP

---

### 8.1 4 Pilastri

---

help

help

help



## 8.2 Astrazione

---

help

help

help

## 8.3 Incapsulamento

---

help

help

help

## 8.4 Ereditarietà

---

help

help

help

## 8.5 Polimorfismo

---

help

help

help

## 9. Relazioni

---

help

help

help

## 10. SOLID

---

### 10.1 SOLID

---

help

help

help

## 10.2 Principio di responsabilità esclusiva

---

help

help

help

## 10.3 Principio aperto/chiuso

---

help

help

help



## 10.4 Principio di sostituzione di Liskov

---

help

help

help

## 10.5 Principio di segregazione dell'interfaccia

---

help

help

help

## 10.6 Principio di inversione delle dipendenze

---

help

help

help

## 11. UML

---

### 11.1 UML

---

help

help

help

## 11.2 Classe UML

---

help

help

help

## 11.3 StateChart UML

---

help

help

help

## 12. Tipi di progettazione per la programmazione PLC

---

help

help

help

## 13. Modelli di progettazione

---

### 13.1 Modelli di progettazione

---

help

help

help



## 13.2 Modello di strategia

---

help

help

help

## 13.3 Modello astratto della fabbrica

---

- [iec-61131-6-abstract-factory-english, stefanhenneken.net](#)

## 14. Links

---

help

help

help