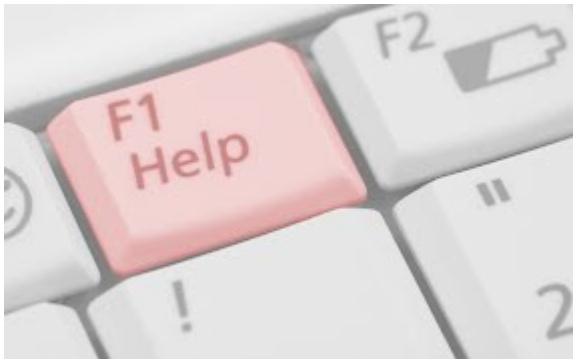
Object Oriented Programming OOP IEC61131-3 Youtube Course by Runtimevic

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

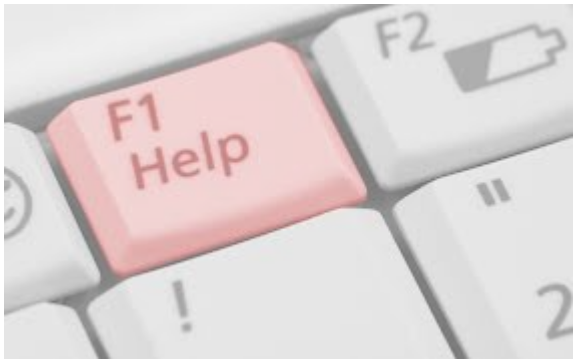
runtimevic

Table of contents

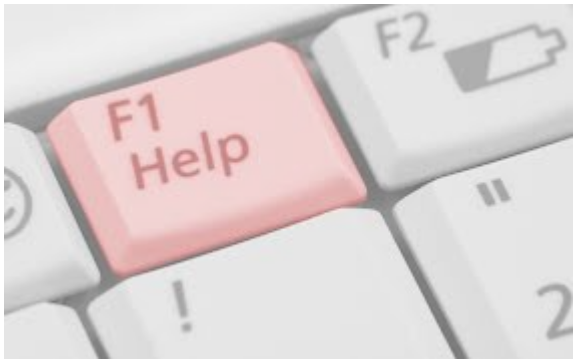
1. Requeriments



2. Introduction

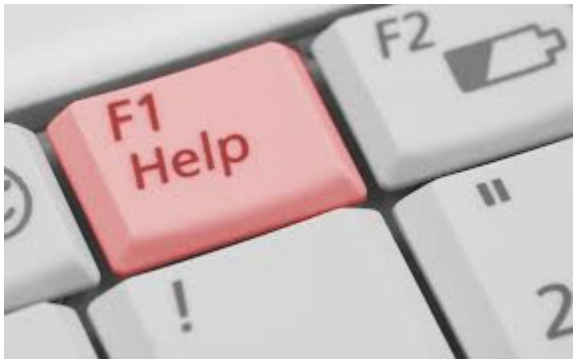


3. Types of paradigms



4. Concepts Previous

4.1 Type of Data



4.2 Variable types and special variables

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.

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 tambien tiene que estar declarada la misma variable con el mismo nombre en una GVL (Lista de Varaibles 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 reinicializan

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 reinicializan

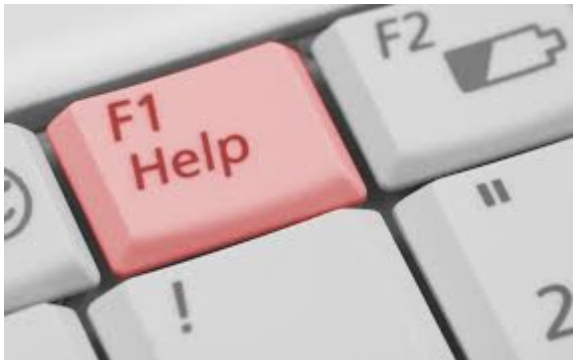
Descargar

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

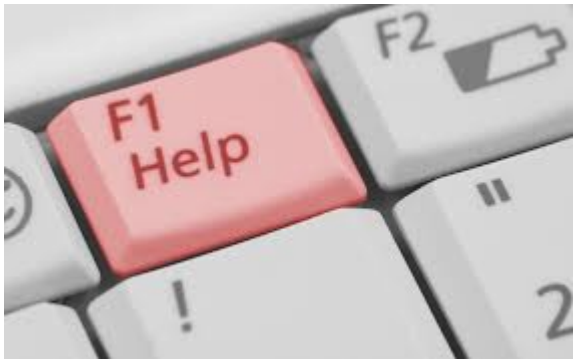
---

- 🔗 infosys.beckhoff.com/
- 🔗 infosys.beckhoff.com/
- 🔗 www.plccoder.com/instance-variables-with-var_inst
- 🔗 www.plccoder.com/var_temp-var_stat-and-var_const
-

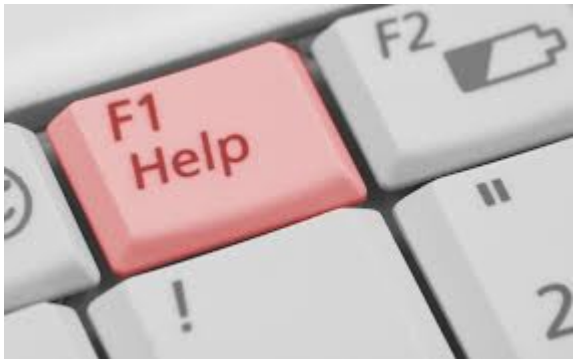
🔗 Tipos de variables y variables especiales



4.3 Access modifiers

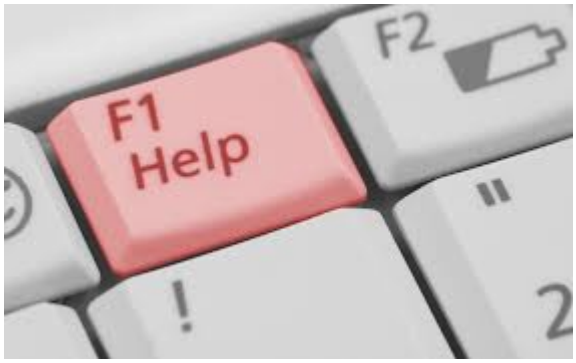


4.4 Access Modifiers Table



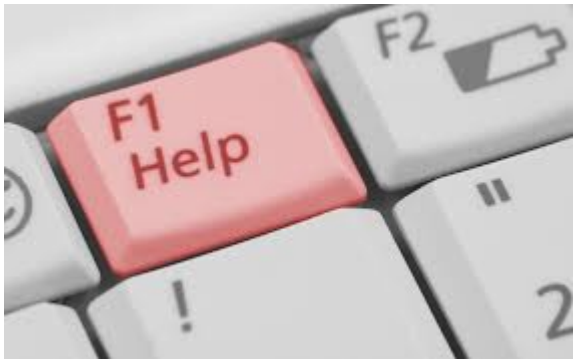
5. Classes and Objects

5.1 Classes and Objects

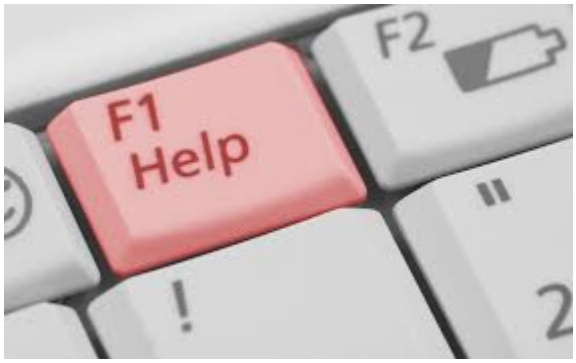


5.2 Function Block

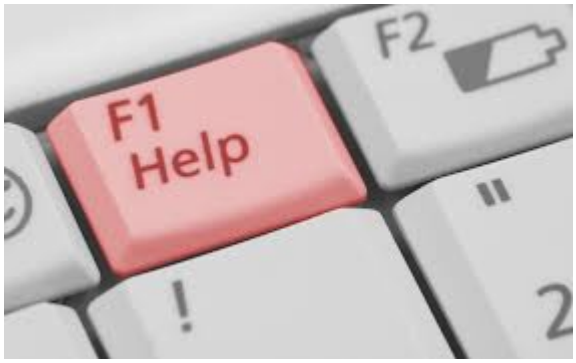
5.2.1 Function Block



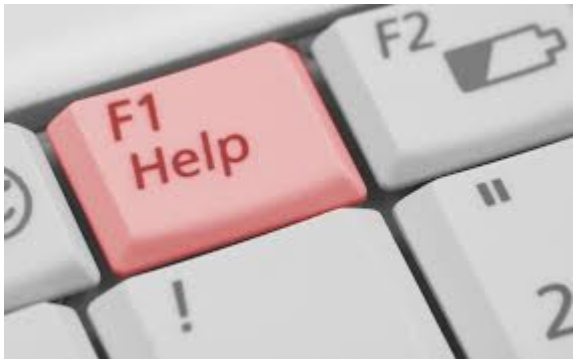
5.2.2 Function Block Access Modifiers



5.2.3 Function Block Declaration variables

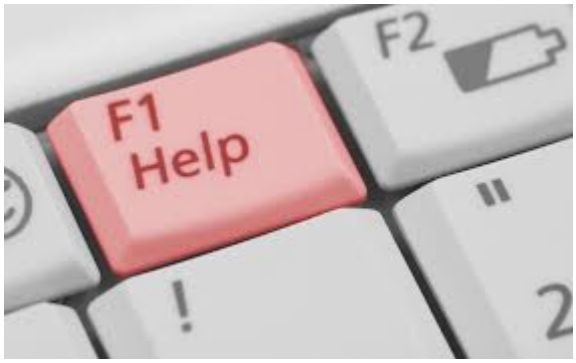


5.2.4 Constructor and Destructor

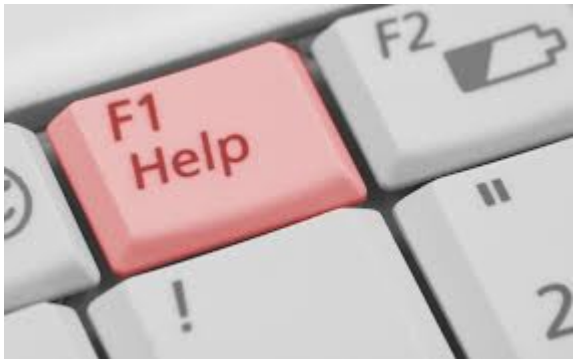


5.3 Object Method

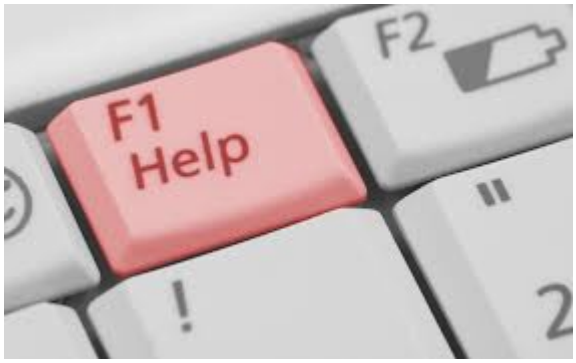
5.3.1 Method



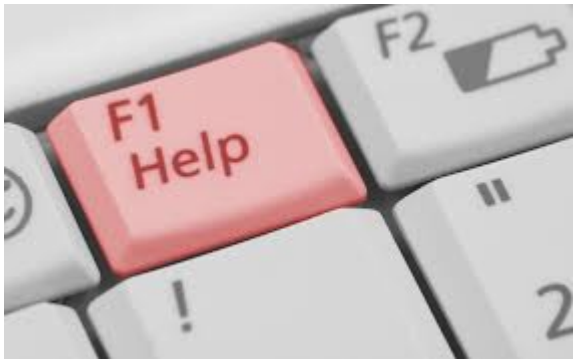
5.3.2 Method access modifiers



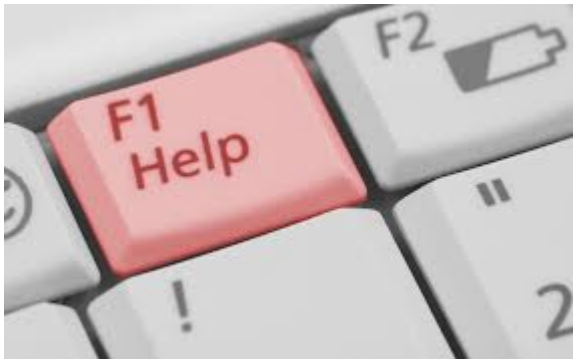
5.3.3 Method Declaration of variables



5.3.4 Method return variable types



5.4 Object Property



5.5 Inheritance

5.5.1 Inheritance Function Block

help

help

help

5.5.2 Inheritance Structure

help

help

help

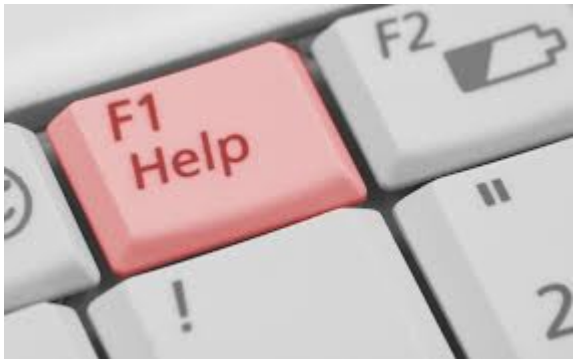
5.5.3 Inheritance Interface

help

help

help

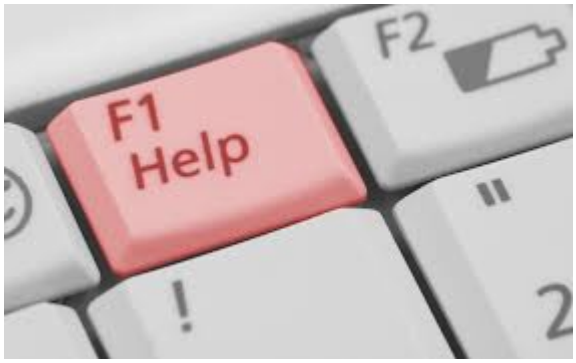
5.6 THIS pointer



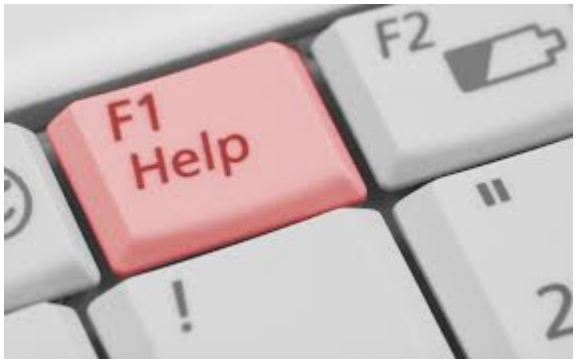
THIS pointer The THIS pointer is available to all function blocks and points to the current function block instance. This pointer is required whenever a method contains a local variable which obscures a variable in the function block.

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.

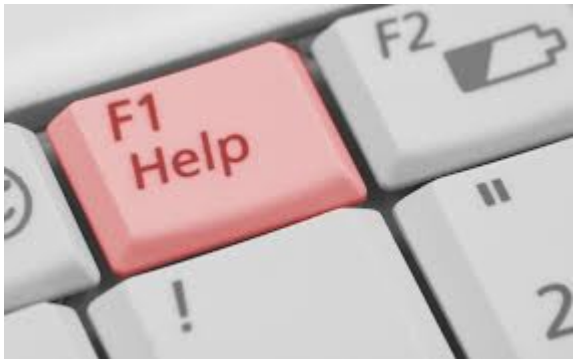
5.7 SUPER pointer



5.8 Interface

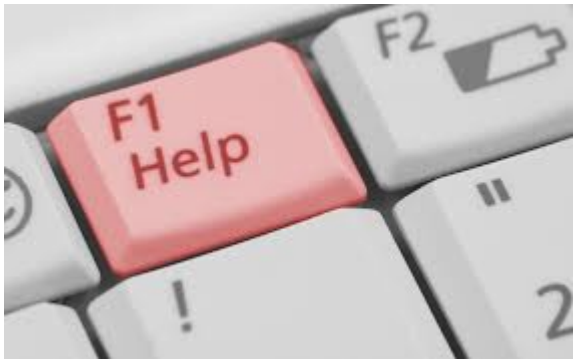


5.9 pointer and reference

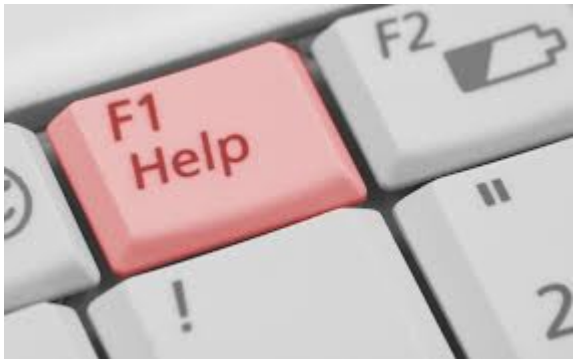


5.10 Keyword Abstract

5.11 Abstract FB vs. Interface

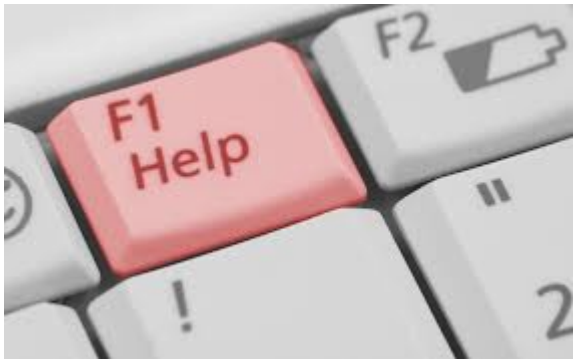


5.12 Fluent Interface

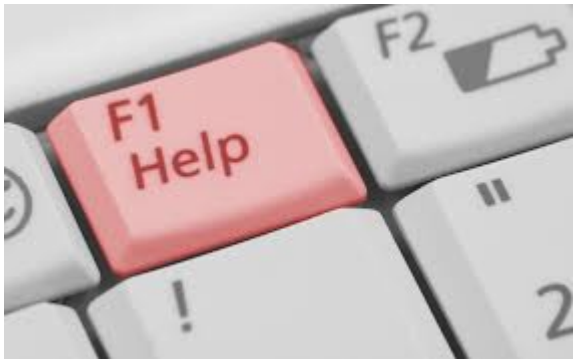


6. OOP Principles

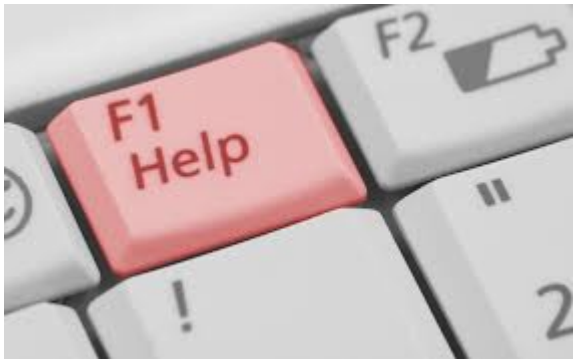
6.1 4 Pillars



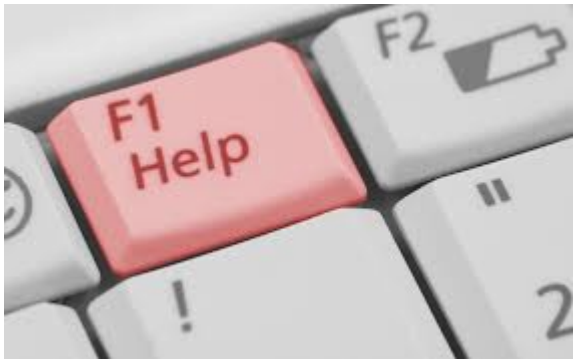
6.2 Abstraction



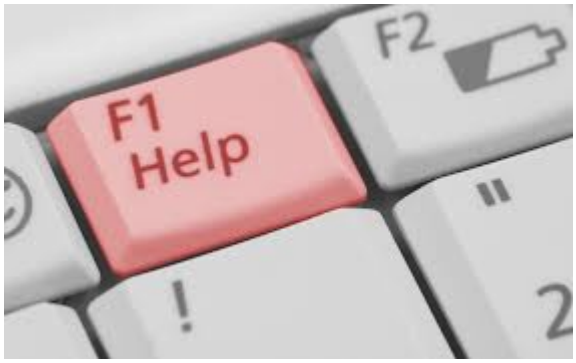
6.3 Encapsulation



6.4 Inheritance

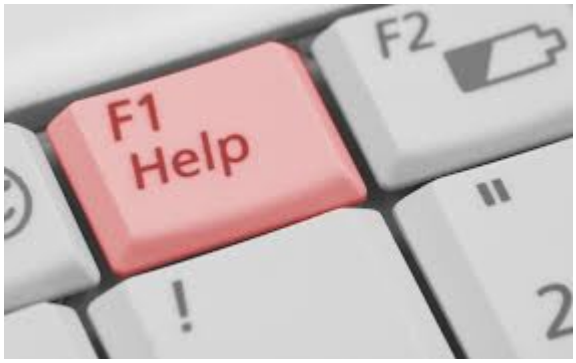


6.5 Polymorphism

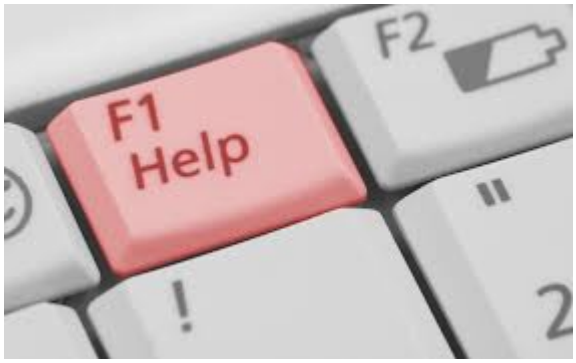


7. SOLID

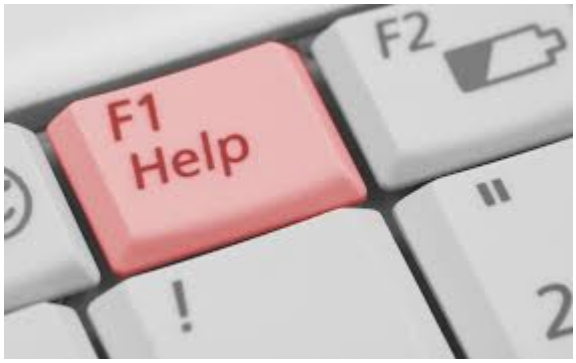
7.1 SOLID



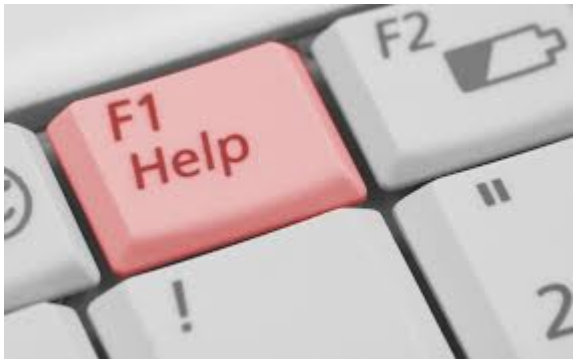
7.2 Single Responsibility Principle - SRP



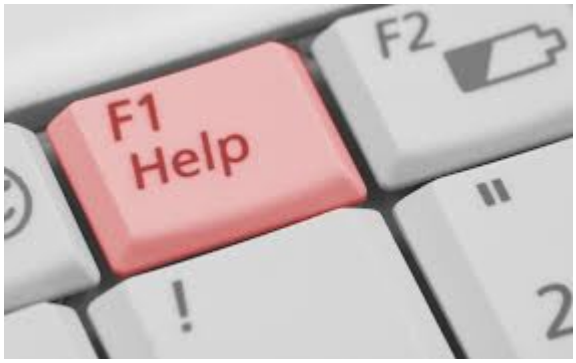
7.3 Open/Closed Principle - OCP



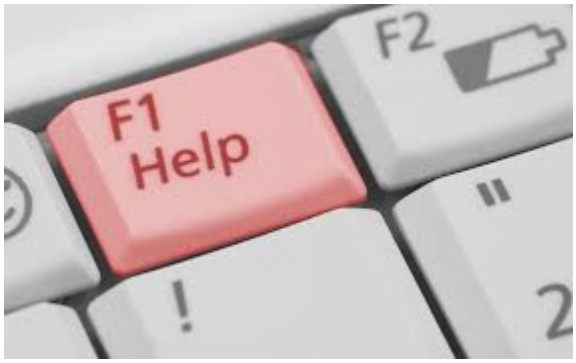
7.4 Liskov Substitution Principle - LSP



7.5 Interface Segregation Principle - ISP

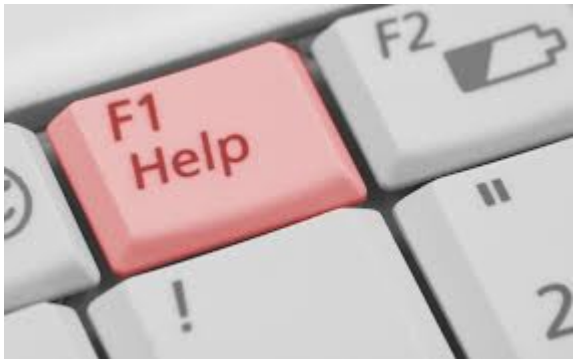


7.6 Dependency Inversion Principle - DIP

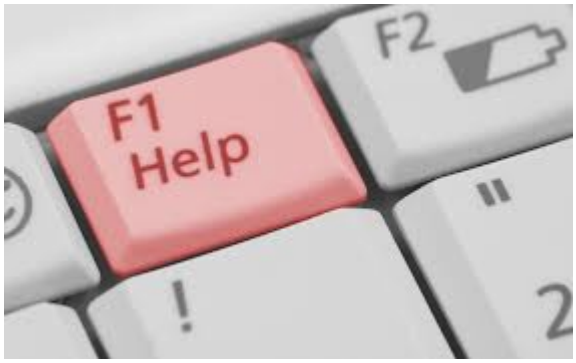


8. UML

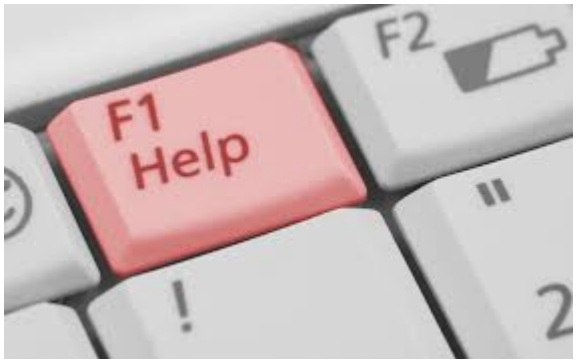
8.1 UML



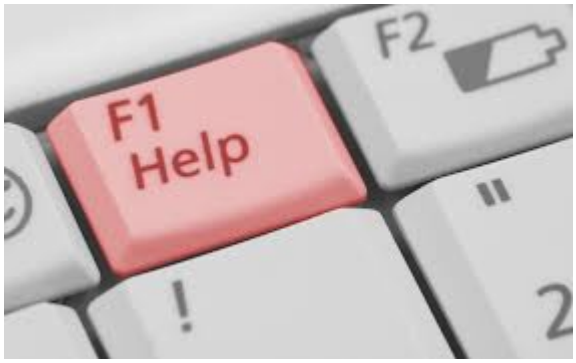
8.2 Class UML



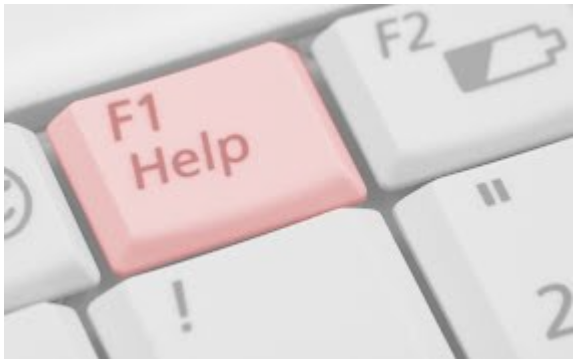
8.3 Relations



8.4 StateChart UML

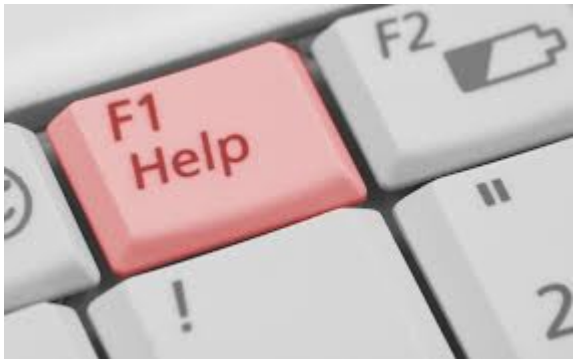


9. Types of Design for PLC programming

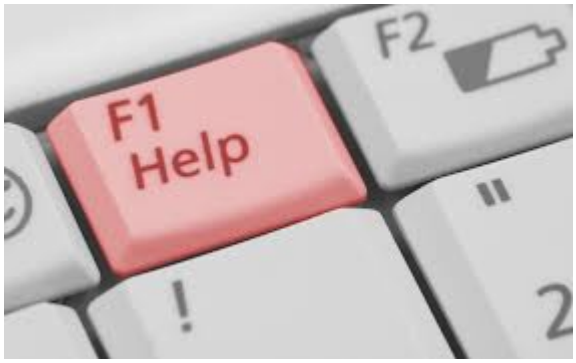


10. Design patterns

10.1 Design patterns



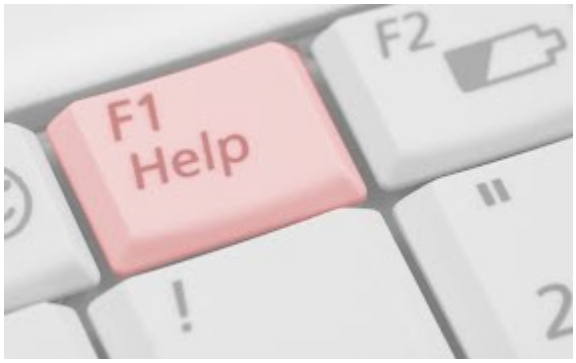
10.2 Strategy Pattern



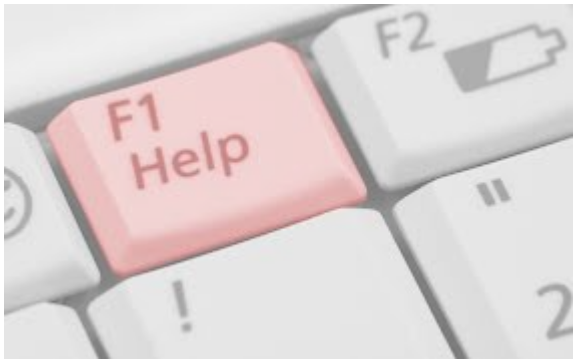
10.3 The Abstract Factory Pattern

- iec-61131-6-abstract-factory-english,stefanhenneken.net

11. Libraries



12. Links OOP



13. TDD

13.1 TDD - Test Drive Development

13.2 Units Test