Нина Кузьмина

Объектно-ориентированное программирование в стандарте МЭК 61131-3

Исторически развитие языков программирования ПЛК проходило своим уникальным путём. Первый программируемый контроллер Modicon 084 имел всего 4 кбайт памяти и программировался с помощью языка, похожего на релейно-контактные диаграммы [1]. Такой подход позволил специалистам, работающим в системах автоматизации, плавно и без больших затрат на переобучение и техподдержку перейти от использования реле и контактов к применению ПЛК.

С развитием технологий системы автоматизации и используемое оборудование усложнялись, появлялись новые производители, поэтому перед сообществом автоматизаторов встал вопрос о стандартизации подходов и языков программирования, для того чтобы ускорить и систематизировать работу с ПЛК, а также обеспечить возможность переноса проектов между платформами разных производителей.

Самым первым стандартом, описывающим независимые от производителя языки программирования для промышленной автоматизации, стал МЭК 61131-3.

На момент создания стандарт выполнил возложенные на него функции: он обеспечил открытость систем, стандартизировал и описал языки программирования ПЛК [2]. Главной особенностью стандарта МЭК стало то, что он включал в себя не один язык программирования, а пять: два текстовых (ST и IL) и три графических (FBD, LD, SFC) [3]. Пользователь мог выбирать язык, который удобен для него и для конкретной задачи. По мере развития данный стандарт стал одним из самых распространённых стандартов программирования ПЛК [4]. Однако при этом многие подходы подвергались критике, так как не всегда стандарт успевал за бурно развивающимися технологиями и возможностями современных систем.

По мере того как программы для ПЛК становились всё сложнее и запутаннее, вставал вопрос о том, как сделать программирование более быстрым, удобным и соответствующим современным подходам. Одним из новшеств, позволяющих сделать это, стало введение объектно-ориентированного программирования (ООП) в третью версию стандарта МЭК 61131-3. Самым первым программным комплексом для прикладного программирования ПЛК и встраиваемых контроллеров, внедрившим поддержку объектно-ориентированного программирования в языках стандарта МЭК 61131-3, стал CODESYS V3 [4]. В настоящее время к поддержке среды программирования CODESYS V3 приходят многие производители. Так, в этом году вышел программируемый логический контроллер CPM723-01, дополняющий линейку распределённой системы управления FASTWEL I/O (рис. 1), который программируется в среде CODESYS V3.5 [5].

Рис. 1. Программируемый логический контроллер CPM723

Для того чтобы понять, как используется ООП при программировании ПЛК, рассмотрим работу с объектами на базе данного программного комплекса.

Концепция объектно-ориентированного программирования

Парадигма объектно-ориентированного программирования сформировалась достаточно давно. Первым языком программирования, использовавшим концепцию ООП, стал Симула (Simula), появившийся в 1967 году [6].

Главной идеей ООП является то, что любая проблема может быть разложена на составные части, каждая из которых становится самостоятельным объектом. Объект представляет собой структурированную переменную, которая содержит всю информацию о некотором физическом или абстрактном предмете [7].

Одним из ключевых понятий парадигмы является *класс*. Класс – это элемент программы, который описывает абстрактный тип данных и его частичную или полную реализацию. Класс представляет собой описание множества однотипных объектов. Для понимания концепции ООП рассмотрим в качестве примера промышленный вентилятор, который будет представлять собой класс.

Классы состоят из множества *свойств* и *методов*. Свойства – это набор конкретных параметров, которыми можно охарактеризовать объект класса. Например, класс вентиляторов (рис. 2) будет иметь свойство *статус* вентилятора, которое характеризуется двумя состояниями: включён или выключен.

Кроме свойств, над объектами каждого класса могут совершаться определённые действия, называемые *методами*. Методы – это функции, которые позволяют выполнять какие-либо действия над свойствами класса. В случае вентилятора методом может быть включение и выключение вентилятора. Результатом вызова метода будет являться смена его статуса.

После того, как класс описан, создаётся *объект*, который характеризует конкретный экземпляр данного класса. Например, в случае вентиляторов это может быть определённый вытяжной

Рис. 2. Структурная схема класса, описывающего промышленные вентиляторы

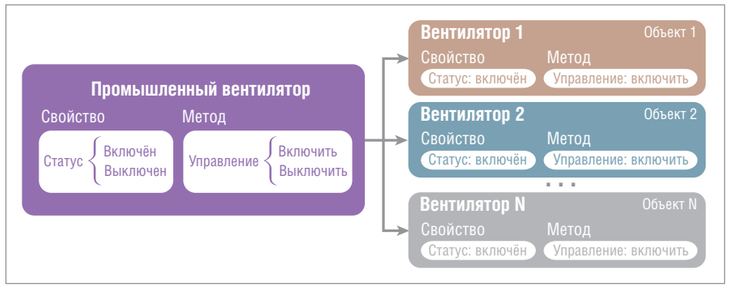

Рис. 3. N объектов (экземпляров), относящихся к классу вентиляторов

вентилятор, располагаемый в конкретной приточно-вытяжной установке. Таким образом, в случае если у нас имеется несколько приточных установок с количеством N вентиляторов, то для каждого из них в программе создаётся свой объект (рис. 3).

Рассмотрим создание классов в среде CODESYS V3. Для описания используемых в классах методов и свойств используются так называемые интерфейсы (Interface, рис. 4). Понятие интерфейса схоже с функциональным блоком, но в отличие от последнего интерфейс не имеет реализации и объявления локальных переменных.

Класс вентиляторов будет иметь одно свойство – статус Status (рис. 5). И для управления статусом используются два метода: запросить состояние (GetStatus) и установить состояние (SetStatus). Для управления вентилятором будет использоваться интерфейс с именем Work, который имеет два метода: включение (Start) и выключение (Stop).

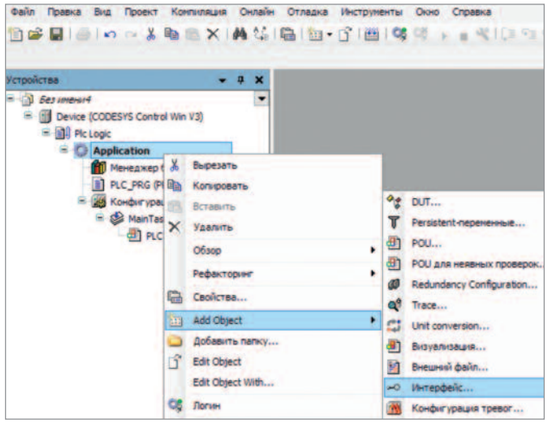

Рис. 4. Создание интерфейса класса

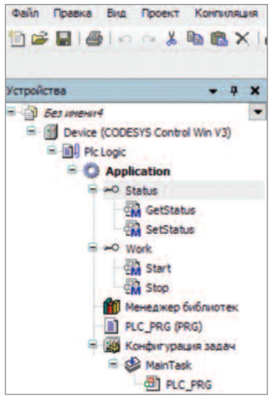

Рис. 5. Интерфейсы, определяющие работу класса

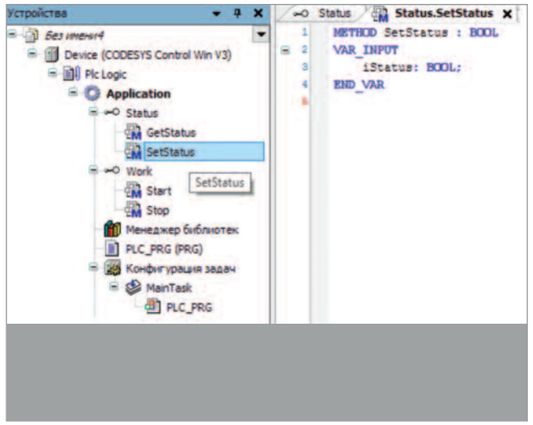

Рис. 6. Реализация метода SetStatus, имеющего возвращаемый тип BOOL

После создания интерфейсов можно переходить непосредственно к созданию класса объектов. Для этого используется программная единица POU (Program Organization Unit) – функциональный блок (FB, Function Block).

Во второй реализации стандарта МЭК 61131-3 функциональный блок использовался в качестве умного определяемого пользователем типа [8]. FB объединял фрагмент программы с данными, которые необходимы для выполнения этого кода, и позволял использовать его много раз (создавать экземпляры). В третьей версии стандарта МЭК существовавшая ранее концепция функционального блока была расширена методами, свойствами и поддержкой наследования классов, что позволяет использовать FB в качестве класса [9].

Итак, на базе FB создаётся класс вентиляторов Fan, который имеет свойства и методы, реализованные в интерфейсах Status и Work (рис. 7).

Поля «Язык реализации метода» и «Язык реализации» позволяют выбрать языки стандарта МЭК, на которых будут реализованы методы и сам класс (или FB).

После добавления функционального блока мы имеем созданный класс, в котором присутствуют описанные нами ин-

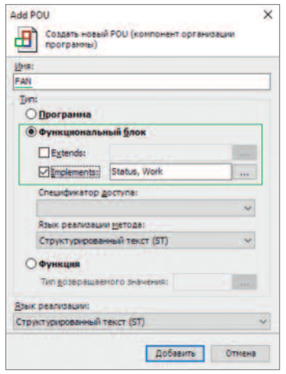

Рис. 7. Поле Implements позволяет добавлять интерфейсы, реализованные в данном классе

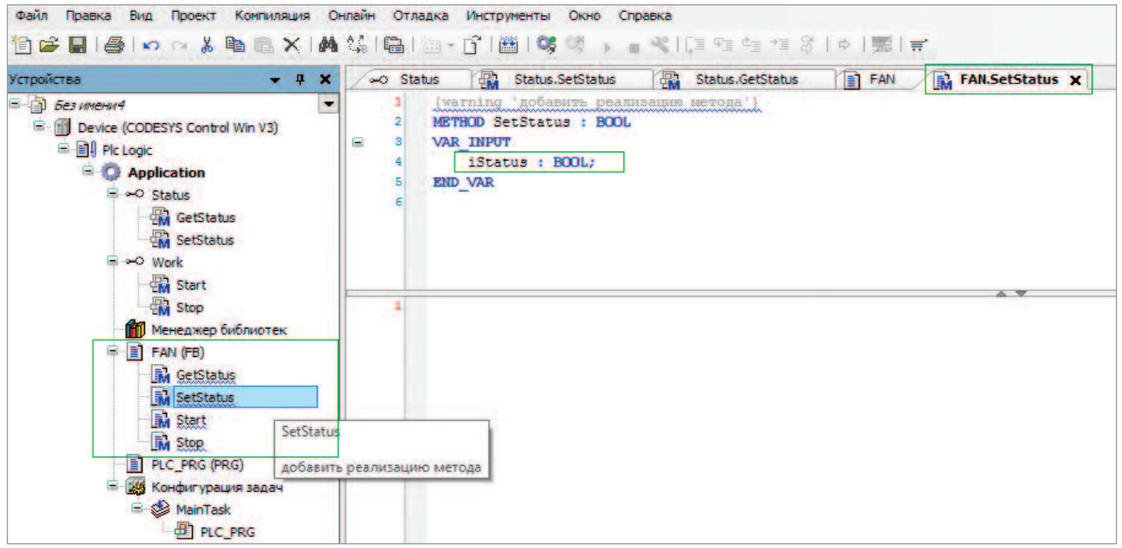

Рис. 8. Созданный класс вентиляторов Fan (методы, созданные в интерфейсе, автоматически перенесены в этот класс)

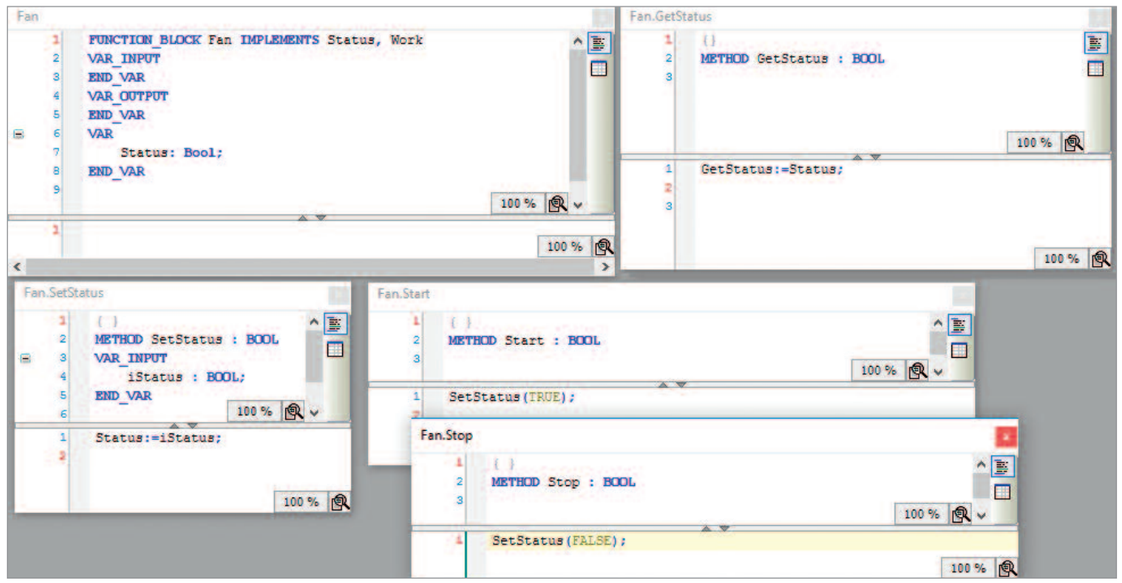

Рис. 9. Реализация методов класса Fan

терфейсы GetStatus, SetStatus, Start, Stop, но ещё не реализованы сами методы работы класса (рис. 8).

На рис. 9 показана реализация класса Fan. Функциональный блок Fan имеет внутреннюю переменную Status, которая связывает между собой SetStatus и GetStatus и является свойством класса Fan. Метод Fan.Start передаёт значение True в метод Fan.SetStatus (и в свойство Status) и включает вентилятор. Метод Fan.Stop передаёт значение False и выключает вентилятор (рис. 9). Таким образом, класс Fan, описывающий промышленные вентиляторы, был создан в среде программирования CODESYS V3.

Основные концепции ООП

Наличие возможности использования классов и объектов не ограничивает парадигму ООП. Основной особенностью языков ООП является соответствие трём следующим концепциям:

1) инкапсуляция;
2) наследование;
3) полиморфизм.

Инкапсуляция – это механизм, который объединяет данные и код, работающий с этими данными, и защищает их от внешнего вмешательства или неправильного использования. По сути, объединённые код и данные, которые представляют собой объект, для пользователя должны представлять собой «чёрный ящик» [7].

В случае выбранного нами объекта вытяжного вентилятора мы имеем на входе управление его состоянием, а на выходе отображение статуса и исправности вентилятора. Вся логика, выполняемая внутри объекта, остаётся для пользователя закрытой.

```
PROGRAM PLC_PRG
VAR
    FanOn: BOOL;
    FanSwither: BLINK;
    fan1: Fan;
END_VAR
```

```
FanSwither(ENABLE:= TRUE, TIMELOW:=T#3S , TIMEHIGH:=T#3S);
FanOn:=FanSwither.OUT;

IF FanOn THEN
    fan1.Start();
ELSE
    fan1.Stop();
END_IF
```

Рис. 10. Объект fan1, принадлежащий классу Fan

PLC_PRG

Device.Application.PLC_PRG

Выражение	Тип	Значение
FanOn	BOOL	TRUE
FanSwither	BLINK	
fan1	Fan	
Status	BOOL	TRUE

Рис. 11. Объект fan1 имеет одно отображаемое для пользователя свойство

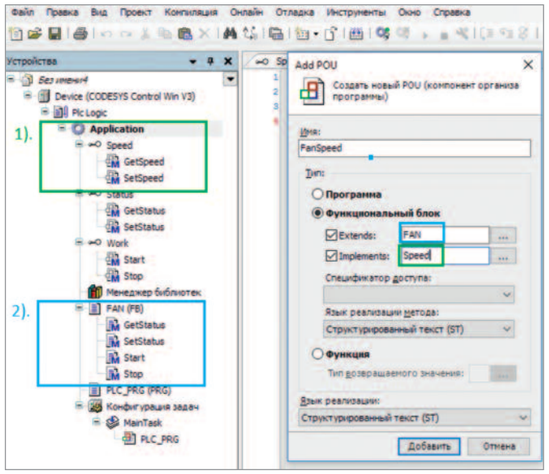

Рис. 12. Создание нового класс FanSpeed

Данный механизм мы можем увидеть при использовании объекта fan1, принадлежащего классу Fan (рис. 10). В программе PLC_PRG, выполняемой контроллером, создаётся объект fan1, принадлежащий классу Fan.

Переменная FanOn управляет включением вентилятора. Генератор импульсов FanSwitcher переключает эту переменную каждые 3 секунды. Методы fan1.Start() и fan1.Stop() включают и выключают вентилятор соответственно (свойство Status).

После загрузки приложения и его запуска мы можем видеть только свойства объекта, а все созданные методы отображаться не будут (рис. 11).

Следующая концепция – *наследование* – позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определённое в другом классе. Наследование позволяет ускорить время разработки программы, так как нет необходимости заново создавать новый класс, можно дополнить уже имеющийся новыми свойствами.

Рассмотрим механизм наследования, используя созданный нами ранее класс Fan. Создадим новый класс промышленных вентиляторов, которые поддерживают также управление скоростью.

Созданный до этого класс Fan возьмём в качестве базового и дополним его новыми свойствами и методами, такими как скорость вентилятора и управление скоростью. Для этого вначале создаётся дополнительный интерфейс Speed (рис. 12), который позволяет устанавливать скорость (SetSpeed) и считывать её (GetSpeed). Далее с помощью добавления нового POU типа FB создаётся ещё один класс FanSpeed. Для того чтобы использовать базовый класс Fan, его необходимо указать в поле Extends (расширение).

Новый класс FanSpeed будет иметь все свойства и методы, присущие классу Fan, поэтому описывать их нет необходимости. Однако добавленный характерный только для него метод управления скоростью необходимо описать (рис. 13). В функциональном блоке FanSpeed в области VAR добавлено новое свойство Speed с типом INT. Метод SetSpeed с помощью входной переменной iSpeed задаёт скорость. Внутри метода реализовано также ограничение скорости между значениями от 0 до 100. Метод GetSpeed считывает значение скорости.

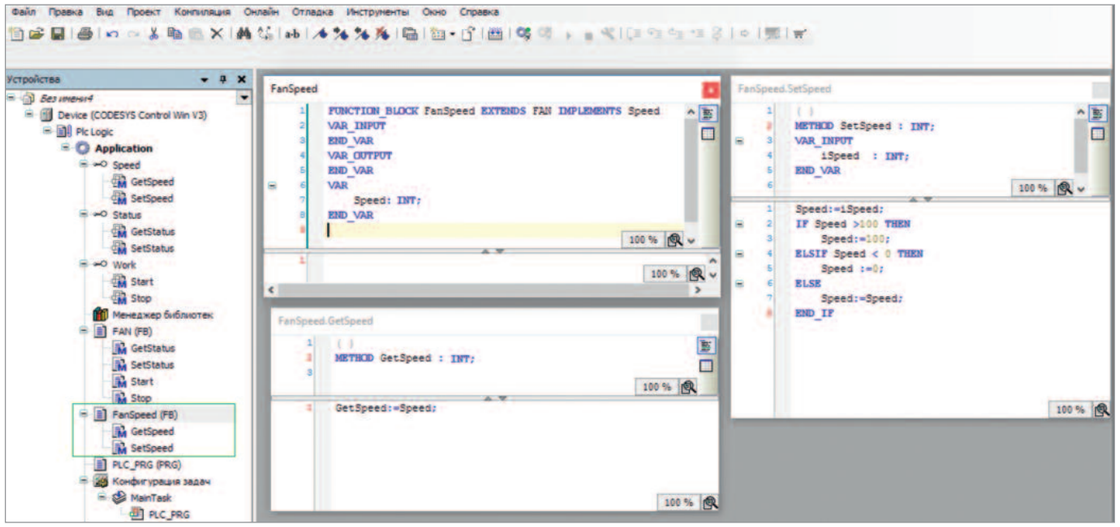

Рис. 13. Реализация нового класса FanSpeed

Объект fan2 (рис. 14), принадлежащий классу FanSpeed, имеет все методы и свойства, которые имеет объект fan1 класса Fan, но в дополнение также имеет скорость. Для передачи значения скорости используется генератор сигнала GEN (встроенная библиотека Util), который генерирует сигнал в виде синусоиды. В режиме online мы видим, что объект fan2 имеет и состояние Status, и скорость (рис. 15).

И последней концепцией, связанной с ООП, является *полиморфизм*. Полиморфизм определяется как возможность объектов с одинаковой спецификацией иметь различную реализацию. Кратко смысл полиморфизма можно выразить фразой «Один интерфейс – множество реализаций» [7]. Полиморфизм в CODESYS V3 реализуется только с использованием интерфейсов. Легче всего понять концепцию полиморфизма на примере класса вентиляторов FanSpeed, который поддерживает управление скоростью. Регулировка скорости может быть как

```
PROGRAM PLC_PRG
VAR
    FanOn: BOOL; (*Кнопка включения вентилятора*)
    FanSwither: BLINK;
    fan1: Fan;
    fan2: FanSpeed;
    SpeedGenerator: GEN ; (*Генератор управляющего сигнала скорости*)
END_VAR
```

```
FanSwither(ENABLE:= TRUE, TIMELOW:=T#3S , TIMEHIGH:=T#3S);
SpeedGenerator(MODE:=SINUS, BASE:=TRUE, PERIOD:=T#10S, AMPLITUDE:=110);

FanOn:=FanSwither.OUT;

IF FanOn THEN
    fan1.Start();
    fan2.Start();
ELSE
    fan1.Stop();
    fan2.Stop();
END_IF

fan2.SetSpeed(SpeedGenerator.OUT);
```

Рис. 14. Код, вызывающий объекты классов Fan и FanSpeed

PLC_PRG

Device.Application.PLC_PRG

Выражение	Тип	Значение
FanOn	BOOL	TRUE
FanSwither	BLINK	
fan1	Fan	
Status	BOOL	TRUE
fan2	FanSpeed	
Status	BOOL	TRUE
Speed	INT	99

Рис. 15. Состояние Status вентилятора fan2 равно TRUE, скорость равна 99

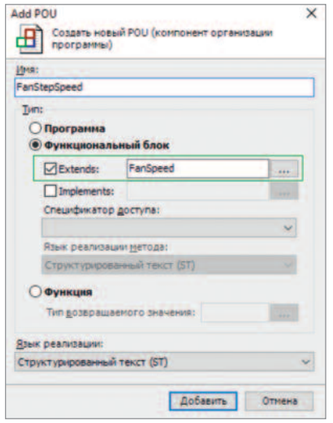

Рис. 16. Класс FanStepSpeed, созданный на базе класса FanSpeed

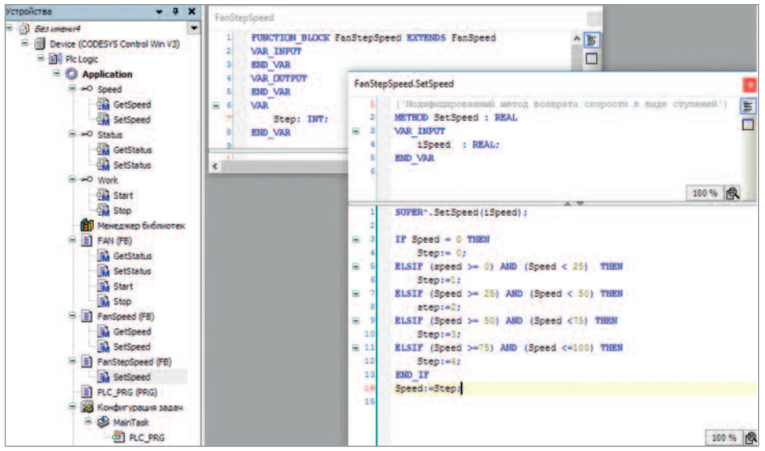

Рис. 17. Преобразование метода SetSpeed

плавной, так и ступенчатой. В классе FanSpeed скорость имеет значение от 0 до 100. Но если нам необходимо, чтобы скорость изменялась ступенями, то для этого нет необходимости заново переписывать метод Speed, достаточно просто его преобразовать.

Новый класс FanStepSpeed в качестве базового использует класс FanSpeed (рис. 16).

Для того чтобы получить скорость в виде ступеней, необходимо преобразовать метод SetSpeed и добавить новое свойство Speed (рис. 17).

В методе SetSpeed строчка SUPER^.SetSpeed(iSpeed) позволяет обращаться к переменным и методам базового класса. Идущий далее код позволяет разбить входящее значение скорости на 4 ступени.

Как показано на рис. 18, объект fan3, принадлежащий к новому классу FanStepSteep, вызывается точно так же, как fan2. Однако в режиме online мы видим, что новый объект имеет дополнительное свойство – Step (рис. 19).

Таким образом, класс FanSpeed был преобразован, и на базе него был создан новый класс вентиляторов.

Заключение

Языки ООП имеют и плюсы, и минусы. Как мы видим, одним из главных минусов ООП является то, что создание классов – довольно трудоёмкий процесс, требующий знания принципов работы описываемого нами объекта.

Однако трудности создания классов окупаются простотой их использования в дальнейшем. Как видно из предложенного примера, в результате применения парадигмы ООП мы получаем более структурированный программный код, позволяющий защитить внутреннюю информацию (инкапсуляция) [8]. Возможность повторного использования и дополнения (наследование), а также изменение (полиморфизм) существующего кода позволяет ускорить процесс программирования. Поэтому, если нам надо использовать сотни однообразных объектов, легче один раз прописать концепцию работы одного элемента, чем много раз писать один и тот же фрагмент программы.

Несмотря на то что некоторые программисты предсказывали смерть данной концепции, как, например, в известной в мире программирования работе Ричарда Гэбриела «Почему объектно-ориентированное программирование провалилось?»[9, 10], ООП продолжает активно развиваться и внедряться в системы.

```
PLC_PRG
1  PROGRAM PLC_PRG
2  VAR
3      FanOn: BOOL; ('Кнопка включения вентилятора')
4      FanSwither: BLINK;
5      fan1: Fan;
6      fan2: FanSpeed;
7      fan3: FanStepSpeed;
8      SpeedGenerator: GEN ; ('Генератор управляющего сигнала скоростью')
9  END_VAR

1  FanSwither(ENABLE:= TRUE, TIMELOW:=T#3S , TIMEHIGH:=T#3S);
2  SpeedGenerator(MODE:=SINUS, BASE:=TRUE, PERIOD:=T#10S, AMPLITUDE:=110);
3
4  FanOn:=FanSwither.OUT;
5
6  IF FanOn THEN
7      fan1.Start();
8      fan2.Start();
9      fan3.Start();
10 ELSE
11     fan1.Stop();
12     fan2.Stop();
13     fan3.Stop();
14 END_IF
15
16 fan2.SetSpeed(SpeedGenerator.OUT);
17 fan3.SetSpeed(SpeedGenerator.OUT);
```

Рис. 18. В программный код добавлен объект fan3, принадлежащий классу FanStepSteep

PLC_PRG

Device.Application.PLC_PRG

Выражение	Тип	Значение
FanOn	BOOL	TRUE
FanSwither	BLINK	
fan1	Fan	
Status	BOOL	TRUE
fan2	FanSpeed	
Status	BOOL	TRUE
Speed	INT	99
fan3	FanStepSpeed	
Status	BOOL	TRUE
Speed	INT	4
Step	INT	4
SpeedGenerator	GEN	

Рис. 19. Объект fan3 имеет новое свойство Step, при скорости 99 оно равно 4

Наличие объектно-ориентированного подхода в среде программирования CODESYS V3 ни к чему не обязывает, но при грамотном использовании существенно упрощает жизнь. ●

Литература

1. Dunn A. The father of invention: Dick Morley looks back on the 40th anniversary of the PLC [Электронный ресурс] // Сайт Manufacturing Automation. – Режим доступа : http://www.automationmag.com/programable-control/features/the-father-of-invention-dick-morley-looks-back-on-the-40th-anniversary-of-the-plc.html.
2. Lydon B. IEC 61131-3 industrial control programming standard advancements [Электронный ресурс] // Сайт ISA. – Режим доступа : https://www.isa.org/standards-publications/isa-publications/intech-magazine/2012/october/system-integration-iec-61131-3-industrial-control-programming-standard-advancements/.
3. Лиференко В. Программирование ПЛК и стандарты IEC 61131-3 // Компоненты и технологии. – 2006. – № 4.
4. Петров И.В. CODESYS – повседневный инструмент программиста ПЛК // Автоматизация в промышленности. – 2012. – № 8.
5. Контроллеры узла сети Modbus TCP (Ethernet) [Электронный ресурс] // Сайт компании FASTWEL. – Режим доступа : http://www.fastwel.ru/products/fastwel-io/programmiruemye-kontrollery-uzla-seti/370451/.
6. Sklenar J. Introduction to OOP in Simula [Электронный ресурс] // Сайт University of Malta. – Режим доступа : http://staff.um.edu.mt/jskl1/talk.html.
7. Буч Г., Максимчук Р.А., Энгл М.У., Янг Б.Дж., Коналлен Дж., Хьюстон К.А. Объектно-ориентированный анализ и проектирование с примерами приложений (UML 2). – М. : Вильямс, 2010.
8. Burleigh A. Object-oriented design: How an update to a PLC programming standard will benefit automation software programmers [Электронный ресурс] // Сайт Manufacturing Automation. – Режим доступа : https://www.automationmag.com/technology/software/3947-object-oriented-design-how-an-update-to-a-plc-programming-standard-will-benefit-automation-software-programmers.
9. Schuenemann U. Programming PLCs with an Object-Oriented Approach [Электронный ресурс] // Сайт CODESYS. – Режим доступа : https://www.codesys.com/news-events/publications/details/article/programming-plcs-with-an-object-oriented-approach-an-article-by-dr-ulf-schuenemann-software-engi.html.
10. Савчук И. Почему объектно-ориентированное программирование провалилось? [Электронный ресурс] // Сайт CITForum.ru. – Режим доступа : http://citforum.ru/gazeta/165/.
11. Gabriel R.P. Objects have failed [Электронный ресурс] // Сайт Dreamsongs. – Режим доступа : http://www.dreamsongs.com/ObjectsHaveFailedNarrative.html.

Автор – сотрудник фирмы ПРОСОФТ
Телефон: (495) 234-0636
E-mail: info@prosoft.ru

НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ

ICONICS – лучший разработчик приложений 2017 года

Компания ICONICS признана партнёром Microsoft 2017 года как лучший разработчик приложений (Application Development Partner).

Компания ICONICS стала победителем среди всех глобальных партнёров компании Microsoft в номинации 2017 Microsoft Application Development Partner. Награду присудили за самые современные инновационные программные решения, разработанные на базе технологий Microsoft. Победители в нескольких категориях были выбраны из более чем 2800 участников из 115 стран мира.

Президент и генеральный директор ICONICS Русс Агруса так прокомментировал событие: «Это высокая честь для ICONICS – получить четвёртую награду Microsoft в качестве партнёра года. Номинация «Лучший разработчик приложений» имеет для нас особое значение, поскольку отражает стремление помочь клиентам с разработкой решений в области управления энергопотреблением, производством, зданиями, чтобы „сделать невидимое видимым“, то есть максимально полно визуализировать ценные для заказчика данные».

ICONICS предлагает программные решения на базе платформы Azure, которые обеспечивают экономию энергии и эффективность производства в рамках концепции Industry 4.0 по всему миру. Благодаря идеологии открытого протокола, которой придерживается компания, её клиенты могут легко интегрировать собственные ИТ-системы в Microsoft Azure. Интуитивно понятная панель инструментов предлагает богатую 3D-визуализацию и уникальные способы отображения данных, без которых невозможны глубокое понимание бизнес-процессов и высокие показатели возврата инвестиций. Используя технологии Интернета вещей и Big Data, ICONICS помогает своим клиентам в решении таких задач, как снижение потребления электроэнергии, сокращение комиссий за выбросы углерода в атмосферу и повышение энергоэффективности бизнеса.

Компания продолжает поставлять инновационные решения благодаря партнёрству с Microsoft и расширяет свою поддержку платформы Azure (IoT Hub, Azure Machine Learning, Power BI и Service Fabric), операционной системы Windows 10 (UWP, HoloLens), технологий SQL Server 2016 и Office 365 (онлайн-приложение SharePoint).

Компания ПРОСОФТ является эксклюзивным дистрибьютором ПО ICONICS на территории России и стран СНГ. ●