**УДК 378:004.43**

# ОСНОВЫ ОБЪЕКТНО ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В ВЫСШЕЙ ШКОЛЕ

**Залогова Л.А.**

*ФГБОУ ВО «Пермский национальный исследовательский университет», Пермь, e-mail: zalogova.la@gmail.com*

**В процедурном программировании данные и процедуры для их обработки не связаны по смыслу, то есть не существует способа защиты данных от неправильного использования. Объектно ориентированное программирование (ООП) предлагает другой подход к разработке программ, в котором данные и методы объединяются в классы и, таким образом, между ними устанавливается связь. На основе классов создаются объекты. В процессе исполнения программы объекты взаимодействуют между собой. В настоящее время знание основ ООП определяет успех во многих областях профессиональной деятельности. В статье излагается структура базового курса по ООП. Основное внимание уделяется базовым понятиям и принципам ООП, которые характерны для различных языков объектно ориентированного программирования. Описывается последовательность, в которой следует вводить новые понятия и методы работы с ними. Особое внимание уделяется наиболее важным и сложным вопросам. Объем предлагаемого материала и методика его изложения ориентированы на студентов младших курсов, начинающих изучать ООП. Единственное требование к освоению курса – владение навыками процедурного программирования. Для решения задач в рамках курса используется язык программирования C#.**

Ключевые слова: классы, объекты, наследование, полиморфизм, интерфейсы, взаимодействие объектов.

# OBJECT-ORIENTED PROGRAMMING BASICS IN A HIGHER SCHOOL

**Zalogova L.A.**

*Perm State University, Perm, e-mail: zalogova.la@gmail.com*

**Data and their processing procedures are not connected in their meaning in procedural programming, because there is no data protection method against their inappropriate application. An object-oriented programming (OOP) presupposes another approach to software design with the data and methods being combined into classes with the links between them. These classes serve to be the basis for the objects. The objects interact in program execution. Today, the success in many professional areas is determined by the knowledge of OOP basics. The paper describes the structure of the course "Introduction into OOP". Key OOP notions and principles which are typical for different object-oriented programming languages are paid special attention to. The paper focuses on the sequence to introduce new notions and methods to work with them. The critical and complicated issues are particularly considered. The scope of the material and course methodology are designed for the juniors who start dealing with OOP. The only requirement to take the course is to have skills in procedural programming. C# programming language is used for solving the problems within the course.**

Keywords: classes, objects, inheritance, polymorphism, interfaces, object interaction.

Определенный способ мышления (парадигма) служит основой для создания языка программирования. Существуют разные парадигмы программирования – процедурная, объектно ориентированная, функциональная, логическая. Каждая из парадигм используется для решения определенного класса задач. Некоторые языки поддерживают несколько парадигм, другие же, наоборот, ориентированы на реализацию только одной парадигмы. Для каждого языка программирования одна из парадигм является основной, а другие парадигмы – дополнительными.

Как правило, для начинающих программистов первой парадигмой становится процедурная, так как освоение предмета сопровождается изучением языка Паскаль или C#. В

рамках процедурной парадигмы программист составляет последовательность вызовов процедур (функций) для решения поставленной задачи. При этом любые процедуры (функции) могут использовать любые данные при условии соответствия количества и типов параметров. Такая ситуация возникает в связи с тем, что данные и процедуры (функции) для их обработки не связаны по смыслу; следовательно, невозможно защитить данные от неправильного использования. Кроме того, применение процедурного программирования связано с определенными трудностями для создания больших программных систем (большие программы трудно создавать, отлаживать, сопровождать).

Объектно ориентированное программирование (ООП) [1] является результатом развития процедурного программирования, однако предлагает другой подход к разработке программ. В ООП данные и методы объединяются в классы, то есть между ними устанавливается связь. На основе классов создаются объекты – главные элементы программы. В процессе выполнения программы объекты взаимодействуют между собой, т.е. обмениваются информацией. ООП позволяет также преодолеть трудности при создании сложных программ.

В настоящее время знание основ объектно ориентированного программирования (ООП) определяет успех во многих областях профессиональной деятельности.

Цель исследования: 1) разработка структуры и содержания курса по основам ООП; 2) демонстрация основ ООП на примерах содержательных задач из различных предметных областей; 3) выполнение объектной декомпозиции и реализация взаимодействия объектов для практических задач.

**Материал и методы исследования:** изучение и анализ работ по ООП, педагогический эксперимент, наблюдение, изучение студенческого творчества, анализ опыта преподавания курса по основам ООП.

**Результаты исследования и их обсуждение**

Автором статьи создано учебное пособие «Основы объектно ориентированного программирования на базе языка C#» (издательство «Лань», 2018 г.), в котором в лаконичной и доступной форме представлен базовый курс по ООП. Основное внимание уделяется базовым понятиям и принципам ООП, которые характерны для различных языков программирования. Реализация же этих понятий и принципов демонстрируется с использованием языка C#, одного из современных перспективных языков программирования. Каждая глава завершается кратким изложением основных положений и заданиями из различных предметных областей.

Учебное пособие предназначено для студентов младших курсов, начинающих изучать ООП. Кроме того, пособие заинтересует тех, кто желает освоить основы ООП на базе языка

C#. Единственное требование к освоению курса – владение навыками процедурного программирования.

Далее в статье описывается содержание курса, изложенного в учебном пособии. Порядок изучения тем курса полностью соответствует последовательности глав учебного пособия.

В первой главе рассматриваются основные понятия (объект, атрибут, состояние объекта, поведение, метод), а также делается краткий обзор принципов ООП. В объектно ориентированном программировании любая программа создается как совокупность взаимодействующих объектов. Шаблоны, на основе которых строятся объекты, называются классами. Согласно первому принципу ООП – *инкапсуляции* - в классе определяются данные и действия, выполняемые над этими данными. Инкапсуляция позволяет поместить данные и методы в класс и таким образом установить связь между ними. Для наглядного представления классов и объектов используется их графическое представление – диаграммы. Классы, как правило, образуют иерархию. Второй принцип ООП – *наследование* - позволяет создать общий (базовый) класс, который может быть унаследован другими более специализированными (производными) классами. В производном классе определяются только такие элементы, благодаря которым он становится более специфическим. И, наконец, третий принцип ООП – *полиморфизм* - означает, что методы с одним и тем же именем могут иметь разный код. Полиморфизм дает возможность работать с объектом, о котором неизвестно, к какому классу он принадлежит – базовому или производному.

Вторая глава содержит описание базовых конструкций C#, необходимых для написания программ (структура консольного приложения, значимые и ссылочные типы, методы преобразования типов, операции и выражения, управление, работа с массивами значений). Более подробную информацию можно найти в [2-4]. На начальном этапе изучения языка лучше всего воспользоваться консольными приложениями, не отвлекаясь на изучение особенностей графического интерфейса. Такой подход позволяет сосредоточиться на основных конструкциях языка, которые в дальнейшем будут использоваться для создания графических приложений.

Классы и объекты – базовые понятия ООП. В первой главе эти понятия рассматриваются очень кратко. В третьей же главе структура классов и объектов излагается более подробно. Особое внимание уделяется вопросам внутренней организации классов (полям, методам, спецификаторам доступа, свойствам), а также созданию и удалению объектов. Для более глубокого понимания материала изложение сопровождается наглядными графическими схемами. Рассматриваются различные способы инициализации

объектов, наиболее интересным из которых является использование конструктора – метода, который инициализирует объект при его создании.

В ряде случаев необходимо решать подобные задачи с разным набором параметров. Объектно ориентированные языки предоставляют возможность выполнять перегрузку методов, то есть. в одном классе определять несколько методов с одинаковыми именами, но разными наборами параметров и реализацией. В связи с этим особое внимание уделяется перегруженным конструкторам, которые позволяют инициализировать объекты по-разному.

В программах достаточно часто встречаются массивы объектов. Важно обратить внимание на процесс создания таких массивов, так как он отличается от создания массивов значений. В остальных же случаях работа с массивами объектов аналогична работе с массивами значений. Кроме того, полезно изучить коллекции – упорядоченные наборы произвольного количества элементов. Работа с коллекциями отличается от работы с массивами. Память под коллекции не фиксируется. Размер коллекции увеличивается и уменьшается в соответствии с количеством ее элементов. Методы добавления и удаления элементов реализованы внутри коллекции. Однако массивы занимают меньше места в памяти, и работа с ними ведется быстрее, чем с коллекцией.

Так как методы класса, как правило, имеют параметры, излагается и демонстрируется на схемах передача параметров значимых и ссылочных типов – по значению и по ссылке. Передача параметров значимых типов отличается от передачи параметров ссылочных типов.

При подстановке значением параметра значимого типа нет никакого способа изменить значение соответствующего фактического параметра и, следовательно, передать через него результат работы функции. Чтобы изменить значение такого параметра, нужно воспользоваться передачей параметра по ссылке. При передаче значением параметра ссылочного типа в область данных вызываемой функции записывается ссылка, например ссылка на объект. Память под сами объекты выделяется в момент их создания в специальной области памяти – куче. Обращение к объекту выполняется через ссылку, поэтому изменение состояния объекта в вызываемой функции изменяет его состояние в вызывающей функции. Таким образом, если ссылка на массив передается методу по значению, то метод может не только обращаться к элементам массива, но и изменять их. При передаче же ссылки на объект по ссылке в вызывающей функции меняется значение самой ссылки.

Возможна ситуация, когда программа содержит несколько методов, не связанных с каким-либо объектом, но реализующих логически связанные функции. Существуют также переменные, общие для всех объектов одного класса. В связи с этим описываются и демонстрируются на примерах правила для работы с такими полями и методами (статическими).

В четвертой главе рассматривается описание и использование наследования, а также его особенности и достоинства. При наследовании базовый класс определяет элементы, характерные множеству других классов. Таким образом, новые классы можно создавать на основе существующего класса-предка. Это, в свою очередь, позволяет избежать дублирования кода в производных классах. Следовательно, уменьшается объем программного кода и сокращается время для его написания. В многоуровневом наследовании производный класс наследует базовому классу, а затем сам становится базовым. В этом случае иерархия содержит несколько уровней. Обоснование необходимости использования многоуровневой иерархии и ее построение описано на примере задачи из конкретной предметной области. Для наглядности многоуровневые иерархии классов представлены в графическом виде.

Создание и использование производных классов требует знания принципов их организации. Дело в том, что объектно ориентированные языки позволяют создавать защищенные элементы, которые доступны для своей иерархии, но закрыты вне этой иерархии. Кроме того, важно понимать особенности конструкторов производного класса. Конструкторы не наследуются. Сначала конструктор производного класса вызывает конструктор базового класса, который создаёт часть объекта, соответствующую базовому классу. Только после этого выполняется создание части объекта, соответствующей производному классу. Если конструктор описан в базовом классе, то и в производном классе конструктор обязательно должен присутствовать.

Рассматриваются также абстрактные классы. Это связано с тем, что в некоторых иерархиях базовый класс является настолько обобщенным, что создание объекта такого класса не имеет смысла; в этом случае попытка создать объект абстрактного класса приведет к сообщению об ошибке во время компиляции.

Пятая глава посвящена реализации полиморфизма в рамках иерархии классов. Обсуждаются особенности строгой типизации и возможность использования нескольких версий одного метода в иерархии классов.

В языках со строгой типизацией нельзя применять операцию к операндам различных типов. Следовательно, ссылка на объект одного класса не может быть использована для ссылки на объект другого класса. Однако существует исключение из этого правила для иерархий классов: ссылке на объект базового класса можно присвоить ссылку на объект производного класса.

В иерархии классов может существовать несколько вариантов описания одноименного метода. Это означает, что метод сначала определяется в базовом классе, а затем переопределяется в производных классах. Интерес представляют ситуации, когда

любая из версий переопределенного метода вызывается посредством ссылки на объект базового класса. В связи с этим возникает вопрос: если обращение к переопределенному методу выполняется через ссылку на объект базового класса, какое описание метода будет использоваться – из базового или производного класса? Здесь следует рассмотреть два случая: использование обычных и виртуальных методов. Если вызывается обычный метод, то тип ссылочной переменной определяет, какие элементы являются доступными. В этом случае будет вызван метод базового класса, то есть установка соответствия между вызовом метода и его описанием выполняется на этапе компиляции (раннее связывание). В случае же использования виртуальных методов и методов-заменителей тип объекта, на который указывает ссылка, определяет, какая версия метода будет выполнена. Поэтому возможен вызов как метода базового класса, так и метода производного класса, то есть решение о том, какую версию метода вызвать, принимается динамически – во время выполнения программы (динамическое связывание). Под «полиморфизмом» здесь понимается возможность существования нескольких (множества) версий (форм) метода в рамках иерархии классов и вызов подходящей версии посредством динамического связывания.

Нередко возникают ситуации, когда невозможно описать реализацию метода в базовом классе. Однако в производных классах этот метод должен быть обязательно реализован. Поэтому необходимо средство, благодаря которому базовый класс заставит производные классы обязательно определить все необходимые методы. Таким средством являются абстрактные методы, которые автоматически являются виртуальными. Поэтому на примере содержательных задач демонстрируется реализация полиморфизма именно для подобных случаев.

В шестой главе рассматривается использование интерфейсов для решения практических задач. Термин «интерфейс» имеет разный смысл в зависимости от контекста. Пользовательский интерфейс определяет способ взаимодействия пользователя с программами. В ООП интерфейсы понимаются в другом смысле: интерфейс определяет набор методов, которые могут быть реализованы классами. Интерфейс не реализует методы, он только определяет, что должно быть сделано, а не как это необходимо сделать. Каждый класс может определить собственную реализацию методов интерфейса. Таким образом, интерфейсы позволяют добавить поведение к классам, т.е. задать их дополнительные возможности.

Полезными в использовании являются стандартные интерфейсы, которые создаются разработчиками языков. Наглядным примером в этом случае являются интерфейсы, предназначенные для сортировки. Дело в том, что объекты коллекции можно упорядочивать по значению разных полей. Поэтому методу сортировки необходимо передать

соответствующую информацию. К примеру, в языке C# класс `List` содержит метод `Sort`, который используется для сортировки объектов различных классов по разным признакам. При этом класс, которому принадлежат сортируемые объекты, должен реализовывать интерфейс `IComparable <T>`. Этот интерфейс содержит единственный метод `CompareTo`, который возвращает результат сравнения двух объектов. Метод `Sort` класса `List` использует метод `CompareTo`, реализация которого зависит от конкретного признака сортировки.

В седьмой главе описывается работа с текстовыми и бинарными файлами. Во всех примерах, рассмотренных в предыдущих главах, использовались средства консольного ввода/вывода. Однако данные, введенные с клавиатуры и отображенные на экране, доступны лишь во время выполнения программы; по завершении работы программы все данные теряются. Содержимое же файлов можно использовать многократно. Кроме того, использование файлов позволяет работать с большими объемами данных.

При попытке обращения к несуществующему файлу во время выполнения программы возникает ошибка - исключительная ситуация (исключение). Язык C# позволяет обработать такую ситуацию, а именно, определить блок кода, который будет автоматически выполняться при возникновении ошибки. Поэтому для организации работы с файлами важно использовать механизм обработки исключений.

Сохранение объектов в текстовом файле требует достаточных усилий, так как в этом случае необходимо отдельно запоминать каждое поле объекта. Чем сложнее объект, тем больший объем кода нужно написать. Ситуация осложняется тем, что объект может содержать унаследованные поля и вложенные объекты. Альтернативой сохранения объектов в текстовом файле является сериализация, т.е. сохранение состояния *объектов* в бинарном файле (последовательности битов, которая интерпретируется только программами). Восстановление состояния объектов, хранимых в бинарных файлах, выполняется в результате десериализации. Сериализация и десериализация значительно сокращают объем кода, который необходимо писать программисту, так как основную работу по реализации сложных процессов сохранения объектов выполняет программное обеспечение. Подробно описывается и демонстрируется на примерах последовательность действий для процессов сериализации и десериализации отдельных объектов, а также массивов и коллекций.

Организация обмена информацией между различными объектами – важнейший аспект ООП. Объекты не существуют изолированно друг от друга. Любая программная система, как правило, реализует взаимодействие объектов, т.е. обмен информацией между ними. Объекты взаимодействуют между собой посредством сообщений (вызовов методов). Особое внимание должно быть уделено организации обмена информацией между методами

различных объектов. Обычно в учебной литературе по ООП излагаются основные конструкции языков и примеры их использования. Однако трудно найти содержательные задачи, для которых описывается реализация взаимодействия объектов. Именно это обстоятельство вызывает трудности в преподавании ООП. Поэтому в восьмой главе излагается один из возможных способов организации взаимодействия объектов. На примере практической задачи обосновывается введение каждого класса, подробно рассматривается структура классов, обсуждается необходимость создания конкретных объектов и способы их взаимодействия. Для наглядного представления передачи сообщений используются диаграммы взаимодействия и диаграммы последовательностей [5]. Объекты и их основные связи изображаются на диаграмме взаимодействия. Временной порядок передачи сообщений описывается диаграммой последовательностей.

**Заключение**

Материал, представленный в статье, является основой курса лекций и практических занятий, проводимых автором в течение нескольких лет. Изложение материала сопровождается содержательными примерами из различных предметных областей. Теоретические знания и опыт разработки программ, полученные в результате изучения этого курса, являются основой для самостоятельного углубления знаний в области ООП.

## Список литературы

1. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно ориентированный анализ и проектирование с примерами приложений. М.: Вильямс, 2010. 720 с.

2. Шилдт Герберт. C# 4.0. Полное руководство. М.: Вильямс, 2015. 1056 с.

3. Э. Троелсен, Ф.Джепикс. Язык программирования C# 6.0 и платформа .Net 4.6. М.: Вильямс, 2016. 1440 с.

4. Клаус Микелсен. Язык программирования C#. СПб.: ООО «ДиаСофтЮП», 2002. 912 с.

5. Гради Буч, Джеймс Рамбо, Ивар Якобсон. Введение в UML от создателей языка. М.: ДМК Пресс, 2015. 498 с.