**32. Объектно-ориентированное программирование. Объекты. классы. Наследование. Пользовательские типы данных. полиморфизм и перегрузка**

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Объектно-ориентированная модель программирования представляет собой методику анализа, проектирования и написания приложений с помощью объектов. Используя ООП, программист может моделировать в своём приложении материальные явления реального мира.

ООП значительно упрощает процесс разработки программы, так как свойства и методы объектов подчиняются простейшей модели безопасности. Если необходимо сделать так, чтобы приложение могло свободно обращаться к объекту, то нужно организовать доступ к нему через открытые свойства и методы. Закрытые свойства и методы используются только самим объектом и не могут использоваться в других процедурах.

Объект — это “строительный блок” в ООП-приложении. Такой строительный блок инкапсулирует часть приложения — процесс, порцию данных или какой-то более абстракный объект. Объекты в языке C# создаются из типов, как и хорошо знакомые составные переменные. Для типа объекта в ООП имеется специальное название — класс. Определения классов позволяют создавать объекты — т.е. реальные именованные экземпляры класса. Понятия “экземпляр класса” и “объект” эквивалентны, но термины “класс” и “объект” означают совершенно разные вещи.

Свойства и поля обеспечивают доступ к содержащимся в объекте данным. Эти данные как раз и являются тем, что отличает отдельные объекты друг от друга, поскольку в свой- ствах и полях разных объектов одного и того же класса могут храниться разные значения.

Термином метод принято называть предоставляемую объектом функцию. Методы могут вызываться так же, как и любые другие функции, и так же возвращать значения и прини-мать параметры (функции были рассмотрены в главе 6). Методы применяются для доступа к функциональным возможностям объекта

В ООП имеются **ограничения,** которые определяют, какие именно фрагменты программного кода могут использоваться другими фрагментами. Объекты выполняют свои задачи и взаимодействуют с другими объектами лишь тогда, когда это действительно необходимо.

Хотя ООП требует нового подхода к программированию, оно обладает несомненными удобствами для программистов. Одни и те же принципы, заключённые в программном фрагменте применяются на всех стадиях от анализа и проектирования до написания кода. ООП позволяет адекватно моделировать поведение реальных объектов. Если они были хорошо спроектированы, то появляется возможность многократного использования программы.

**Абстракция данных**

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор значимых характеристик объекта, доступный остальной программе.

**Инкапсуляция**

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Некоторые языки (например, C++) отождествляют инкапсуляцию с сокрытием, но большинство (Smalltalk, Eiffel, OCaml) различают эти понятия.

**Наследование**

Наследование — свойство системы, позволяющее описать новый класс на основе уже

существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

**Полиморфизм подтипов**

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма — параметрический — в ООП называют обобщённым программированием.

**Класс**

Класс является описываемой на языке терминологии исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это **экземпляр** класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

**Объект**

Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

В центре ООП находится понятие *объекта.* Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Классы могут наследоваться друг от друга. Класс-потомок получает все поля и методы класса-родителя, но может дополнять их собственными либо переопределять уже имеющиеся. Большинство языков программирования поддерживает только единичное наследование (класс может иметь только один класс-родитель), лишь в некоторых допускается множественное наследование — порождение класса от двух или более классов-родителей. Множественное наследование создаёт целый ряд проблем, как логических, так и чисто реализационных, поэтому в полном объёме его поддержка не распространена.

Интерфейс — это класс без полей и без реализации, включающий только заголовки методов. Если некий класс наследует (или, как говорят, реализует) интерфейс, он должен реализовать все входящие в него методы. Использование интерфейсов предоставляет относительно дешёвую альтернативу множественному наследованию.