Основные понятия ООП

C# является объектно-ориентированным языком программирования, вследствие чего предварительно будут приведены основные парадигмы ООП.

В связи с проникновением компьютеров во все сферы социума программные системы становятся более простыми для пользователя и сложными по внутренней архитектуре. Программирование стало делом команды, где маленьким проектом считается тот, который выполняет команда из 5–10 специалистов за время от полугода до года.

Основным способом борьбы со сложностью программных продуктов стало объектно-ориентированное программирование (ООП), являющееся в настоящее время наиболее популярной парадигмой.

ООП – методология программирования, основанная на представлении программного продукта в виде совокупности объектов, каждый из которых является экземпляром конкретного класса. ООП использует в качестве базовых элементов взаимодействие объектов.

Объект – именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение.

В применении к объектно-ориентированным языкам программирования понятия объекта и класса конкретизируются.

Объект – обладающий именем набор данных (полей и свойств объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним. Имя используется для работы с полями и методами объекта.

Любой объект относится к определенному классу. В классе дается обобщен- ное описание некоторого набора родственных объектов.

Объект – конкретный экземпляр класса.

В качестве примера можно привести абстракцию дома или его описание (класс) и реальный дом (экземпляр класса или объект). Объект соответствует логической модели дома, представляющей совокупное описание всех физических объектов.

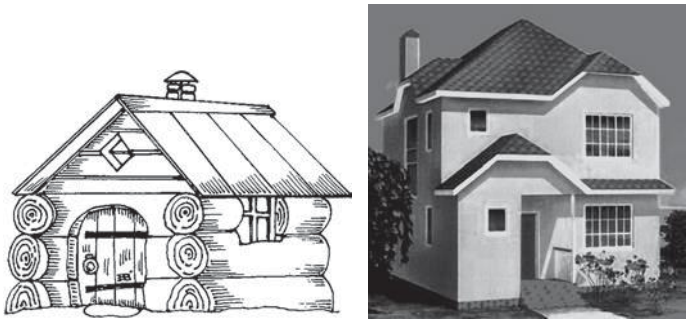

Рис. 2.1. *Описание класса (абстракция) и реальный объект*

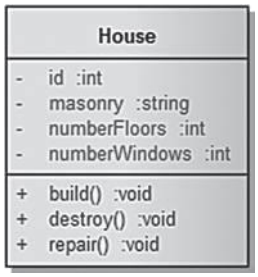

Рис. 2.2. *Графическое изображение класса*

Класс принято обозначать в виде прямоугольника, разделенного на три части. В верхний прямоугольник помещается имя класса, в средний – набор полей с именами, типами, свойствами класса, и в нижний – список методов, их параметров и возвращаемых значений.

Реальный объект должен иметь конкретные значения всех полей, например:

id=35, masonry="brick", numberFloors=2, numberWindows=7.

Объектно-ориентированное программирование основано на принципах:

— инкапсуляции;

— наследования;

— полиморфизма, в частности, «позднего связывания».

Инкапсуляция (encapsulation) – принцип, объединяющий данные и код, манипулирующий этими данными, а также защищающий данные от прямого внешнего доступа и неправильного использования. Другими словами, доступ к данным класса возможен только посредством методов этого же класса.

Наследование (inheritance) – процесс, посредством которого один класс может наследовать свойства другого класса и добавлять к ним свойства и методы, характерные только для него.

Наследование бывает двух видов:

одиночное наследование – подкласс (производный класс) имеет один и только один суперкласс (предок);

множественное наследование – класс может иметь любое количество предков (в Java запрещено).

Полиморфизм (polymorphism) – механизм, использующий одно и то же имя метода для решения похожих, но несколько отличающихся задач в различных объектах при наследовании из одного суперкласса. Целью полиморфизма является использование одного имени при выполнении общих для суперкласса и подклассов действий.

Механизм «позднего связывания» в процессе выполнения программы определяет принадлежность объекта конкретному классу и производит вызов метода, относящегося к классу, объект которого был использован. Механизм «позднего связывания» позволяет определять версию полиморфного (виртуального) метода во время выполнения программы. Другими словами, иногда невозможно на этапе компиляции определить, какая версия переопределенного метода будет вызвана на этапе выполнения программы.

Краеугольным камнем наследования и полиморфизма предстает следующая парадигма: «***объект подкласса может использоваться всюду, где используется объект суперкласса***». То есть при добавлении к иерархии классов нового подкласса существующий код с экземпляром нового подкласса будет работать точно так же, как и со всеми другими экземплярами классов в иерархии.

При вызове метода сначала он ищется в самом классе. Если метод существует, то он вызывается. Если же метод в текущем классе отсутствует, то обращение происходит к родительскому классу и вызываемый метод ищется в этом классе. Если поиск неудачен, то он продолжается вверх по иерархическому дереву вплоть до корня (верхнего класса **Object**) иерархии.