Принципы объектно-ориентированного программирования

Наследование

- В объектно-ориентированных языках программирования классы могут описывать на основе уже существующих классов с наследованием их методов и свойств. При этом один класс может порождать несколько классов-наследников. В то же время, каждый класс может иметь только одного предка. Такое наследование называется последовательным (если бы у класса могло быть несколько предков, то наследование называлось бы параллельным, или множественным).
- Иллюстрацией принципа последовательного наследования может служить класс **Box** («Ящик»), обладающий свойствами **Length**, **Width**, **Height** («Длина», «Ширина», «Высота») и методами их изменения. На основе данного класса можно создать класс-наследник **Garage** («Гараж»), который будет обладать теми же свойствами и методами, что и его предок. Однако у объектов класса «Гараж» будет учтено дополнительное свойство **Material** («Материал») типа **string** и присутствовать метод его задания **SetMaterial(string M)**.
- Точно так же на основе класса **Garage** может быть описан **House** («Дом»), у которого добавятся свойства **RoofHeight** («Высота крыши») и **RoofMaterial** («Материал крыши») и методы их изменения **SetRoofHeight(double H)** и **SetRoofMaterial(string M).**

Инкапсуляция

- Инкапсуляция – принцип объектно-ориентированного программирования, позволяющий скрывать от конечного пользователя устройство и механизм работы объектов класса, оставляя доступ только к средствам управления поведением объекта.
- В роли пользователя может выступать как программист, работающий с объектами в процессе написания программы, так и класс-наследник, использующий для работы с собственными объектами уже имеющиеся свойства и методы класса-предка. В качестве конечного пользователя также может рассматриваться и сам объект, однако у него всегда имеется полный доступ к собственным свойствам и методам.
- Настройка доступа к внутреннему устройству объектов осуществляется при помощи модификаторов доступа **public**, **protected**, **private**.
- Результатом использования механизма инкапсуляции является представление объекта в виде «черного ящика» (капсулы), механизм функционирования которого скрыт, однако для которого имеется возможность подавать на его вход управляющие воздействия и получать на выходе отклик (результат), определяющийся его внутренним устройством.

Полиморфизм

- Принцип полиморфизма позволяет использовать классу-предку методы и свойства классов-потомков.
- В C# все классы наследуются от класса **Object**. В рамках принципа полиморфизма имеется возможность создать переменную класса **Object** и присвоить ей объект любого класса (наследника):

```
Object o = new House();
(House) o.Height = 100;
```

- В данном примере объявляется переменная **o**, которая может содержать ссылку на объект класса **Object**. Тем не менее, мы присваиваем ей ссылку на объект класса **Box**, наследника класса **Object**. В результате мы имеем возможность работать с нашим объектом как с домом, а не как с объектом базового класса. Для этого требуется выполнить приведение типов, указав **(House)** перед заданием действия с переменной. В принципе, благодаря полиморфизму мы можем работать с домом и как с ящиком (ведь дом также является наследником класса **Box**):
  - **(Box) o.Height = 1;**

Наследование от класса Object

- Класс **Object** является предком для всех классов, описываемых в программе. Поэтому если при описании класса не указать, от какого класса он порождается, по умолчанию предком нового класса будет объявлен класс **Object**. Таким образом, следующие две записи абсолютно идентичны:

- ```
  class Person
  {
        // Описание класса
  }
  ```

- ```
  class Person: System.Object
  {
        // Описание класса
  }
  ```

Наследование от класса Object

- Класс **Object** обладает рядом методов. Как следствие, этими методами обладает любой другой класс как наследник класса **Object**. Этими методами являются:
- **Equals**(**Object o**) – метод сравнивает вызвавший его объект с объектом **o**. Если объект-хозяин и объект **o** одинаковы (ссылки содержат адреса на одну и ту же область памяти), то метод возвращает **true**, иначе – **false**.
- **GetHashCode**() – возвращает хеш-код вызвавшего метод объекта.
- **GetType**() – возвращает объект класса **System.Type**, идентичный типу объекта, вызвавшего метод.
- **ToString**() – преобразует класс в строку.
- **Finalize**() – метод вызывается автоматически при высвобождении памяти из-под объекта (уничтожении объекта).
- **MemberwiseClone**() – метод, создает в памяти копию вызвавшего его объекта и возвращает ссылку на эту копию.

Наследование от класса Object

- Для демонстрации работы методов опишем класс **Person**, объект которого хранит имя и фамилию человека.

- ```
  class Person
  {
      public string FirstName { get; set; }
      public string SecondName { get; set; }

      public Person(string firstName, string secondName)
      {
          FirstName = firstName;
          SecondName = secondName;
      }
  }
  ```
  ```

Наследование от класса Object

- Проверим работу метода **ToString()**, объект которого хранит имя и фамилию человека.

- **Person p = new Person(«Имя», «Фамилия»);**
  Console.WriteLine(p.ToString()); // ПространствоИмен.Person

- Метод **ToString()** возвращает строку, содержащую имя пространства имен, в котором был описан класс и имя класса (через точку). Данный метод часто переопределяется программистами под свои задачи вывода информации об объектах класса.