ООП в C#
(наследование классов)

Андрей Голяков

Наследование

Наследование является одним из фундаментальных атрибутов объектно-ориентированного программирования.

Оно позволяет определить дочерний класс, который использует (наследует), расширяет или изменяет возможности родительского класса.

Класс, члены которого наследуются, называется базовым классом. Класс, который наследует члены базового класса, называется производным (дочерним) классом.

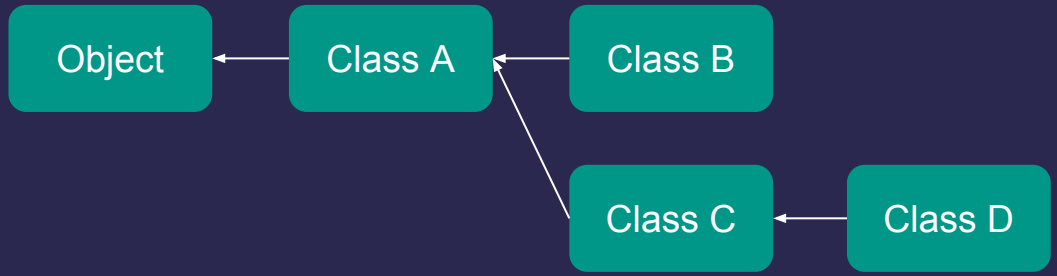
C# и .NET поддерживают только одиночное наследование. Это означает, что каждый класс может наследовать члены только одного класса.

Наследование

Зато наследование может быть выполнено “по цепочке”, т. е., класс D наследуется от класса C, C и B - от класса A.

По умолчанию все классы наследуются от базового класса Object, даже если мы явным образом не устанавливаем наследование. Поэтому все классы кроме своих собственных методов, также будут иметь и методы класса Object:

- ToString()
- Equals()
- GetHashCode()
- GetType()



Наследование

Пример простого наследования

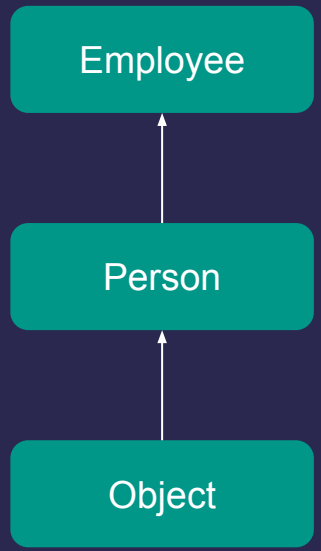
```
// Базовый класс
public class Person
{
  public string Name { get; set; }
}

// Дочерний класс
public class Employee : Person {}

private static void Main()
{
  Person p1 = new Person
  {
    Name = "Andrei"
  };

  Employee e1 = new Employee
  {
    Name = "Sergei"
  };

  Console.WriteLine(p1.Name);
  Console.WriteLine(e1.Name);
}
```



Перекрытие членов базового класса: new

Пример перекрытия членов базового класса

```
// Базовый класс
public class Person
{
  public string Name { get { return "Person"; }
}

// Дочерний класс
public class Employee : Person
{
  new public string Name { get { return "Employee"; }
}

private static void Main()
{
  Person p1 = new Person();

  Employee e1 = new Employee();

  Console.WriteLine(p1.Name);
  Console.WriteLine(e1.Name);
}
```

Самостоятельная работа

Напишите собственный базовый класс BaseDocument, который бы описывал произвольный документ и имел бы:

- Свойства:
  - DocName типа string: наименование документа
  - DocNumber типа string: номер документа
  - IssueDate типа DateTimeOffset: дата выдачи
  - PropertiesString типа string: read-only свойство, формирующее строку для вывода на экран свойств этого класса
- Метод
  - WriteToConsole()

Затем напишите производный класс Passport, унаследованный от BaseDocument, который бы имел дополнительно свойства Country (string) для хранения страны и PersonName (string) для хранения имени владельца.
Напишите новую реализацию свойств PropertiesString и метод WriteToConsole() чтобы произвести сокрытие членов базового класса.
Создайте по одному экземпляру каждого класса в основном потоке программы, инициализируйте их свойства и выведите их на экран используя метод WriteToConsole() соответствующих классов.

Переопределение членов: virtual /override

```
public class Person
{
  public string Name { get; set; }
  public DateTimeOffset DateOfBirth { get; set; }

  public virtual string ShortDescription
  {
    get
    {
      return $"{GetType().Name} " +
             $"name: {Name}, " +
             $"date of birth: {DateOfBirth:dd-MM-yy}";
    }
  }

  public void WriteShortDecription()
  {
    Console.WriteLine(ShortDescription);
  }
}
```

```
public class Employee : Person
{
  public string EmployeeCode { get; set; }

  public DateTimeOffset HireDate { get; set; }

  public override string ShortDescription
  {
    get
    {
      return $"{GetType().Name} " +
             $"code: {EmployeeCode}, " +
             $"name: {Name}, " +
             $"date of birth: {DateOfBirth:dd-MM-yy},"
+
             $"hire date: {HireDate:dd-MM-yy},";
    }
  }
}
```

Самостоятельная работа

Внесите соответствующие изменения в ваши классы BaseDocument и Passport, чтобы заменить сокрытие переопределением.

Удалите ставшую ненужной имплементацию метода WriteToConsole() у дочернего класса Passport.

Наследование конструкторов

Конструкторы не передаются производному классу при наследовании.

При наследовании конструкторов работают следующие правила:

1. Производный и базовый класс могут иметь свои наборы конструкторов
2. В момент создания дочернего объекта вызываются все конструкторы его предков в порядке от самого базового до конструктора текущего класса
3. Если в базовом классе отсутствует конструктор без параметров, необходимо явно вызывать конструктор базового класса, используя ключевое слово base

Наследование конструкторов: base

```
public class Person
{
  public string Name { get; set; }

  public DateTimeOffset DateOfBirth { get; set; }

  public Person(
    string name,
    DateTimeOffset dateOfBirth)
  {
    Name = name;
    DateOfBirth = dateOfBirth;
    Debug.WriteLine(
      "Constructor Person(name, dateOfBirth) called");
  }

  ...
}
```

```
public class Employee : Person
{
  public string EmployeeCode { get; set; }

  public DateTimeOffset HireDate { get; set; }

  public Employee(
    string name,
    DateTimeOffset dateOfBirth)
    : base(name, dateOfBirth)
  {

  }

  public Employee(
    string name,
    DateTimeOffset dateOfBirth,
    string employeeCode,
    DateTimeOffset hireDate)
    : base(name, dateOfBirth)
  {
    EmployeeCode = employeeCode;
    HireDate = hireDate;
  }

  ...
}
```

Самостоятельная работа

Создайте конструкторы для классов BaseDocument и Passport, заполняющие их свойства.

Для конструктора класса Passport не нужно запрашивать параметр docName, нам и так известно, что все документы типа Passport имеют имя “Passport”. Передайте строку “Passport” в качестве строкового литерала при вызове наследуемого конструктора посредством base.

Обновите создание объектов, используя новые конструкторы.

Убедитесь, что код работает как и прежде.

Ссылки на базовый класс

```
Person p1 = new Person("Andrei",
DateTimeOffset.Parse("1982-03-14"));
p1.WriteShortDecription();
// Person name: Andrei, date of birth: 14-03-82

Person e1 = new Employee(
  "Andrei",
  DateTimeOffset.Parse("1982-03-14"),
  "000001",
  DateTimeOffset.Parse("2016-10-01"));

e1.WriteShortDecription();
// Employee code: 000001, name: Andrei,
// date of birth: 14-03-82, hire date: 01-10-16

// Two lines below will raise compilator error
// as Person doesn't know about these fields
//Console.WriteLine(e1.HireDate);
//Console.WriteLine(e1.EmployeeCode);
```

```
var persons = new Person[3];

persons[0] = new Person(
  "Maria", DateTimeOffset.Parse("1987-03-01"));

persons[1] = new Person(
  "Sergey", DateTimeOffset.Parse("1981-12-27"));

persons[2] = new Employee(
  "Andrei",
  DateTimeOffset.Parse("1982-03-14"),
  "000001",
  DateTimeOffset.Parse("2016-10-01"));

foreach (var person in persons)
{
  if (person is Employee)
  {
    // Just cast to Employee and work with it!
    var employee = (Employee) person;
    ...
  }
  else
  {
    ...
  }
}
```

Самостоятельная работа

Добавьте классу Passport метод ChangeIssueDate(DateTimeOffset newIssueDate)

Создайте массив объектов базового класса, наполните его несколькими объектами как базового, так и производного классов.

Перебирая их в цикле и проверяя, является ли класс объекта классом Passport, поменяйте всем паспортам IssueDate на сегодняшнее число.

Выведите на экран информацию о документе используя метод базового класса WriteToConsole().

Домашнее задание

Унаследовать от класса ReminderItem, написанного в прошлой домашней работе 2 класса:

1. PhoneReminderItem
    a. с дополнительным свойством PhoneNumber (string) номер телефона, куда нужно послать сообщение.
    b. с конструктором, который кроме параметров базового класса имеет также дополнительный параметр (для заполнения нового поля)
    c. с переопределенным методом WriteProperties()
2. ChatReminderItem
    a. с дополнительными свойствами
        i. ChatName (string) - название чата
        ii. AccountName (string) - имя аккаунта в чате, которому нужно послать сообщение
    b. с конструктором, который кроме параметров базового класса имеет также два дополнительных (для заполнения новых полей)
    c. с переопределенным методом WriteProperties(), выводящим все свойства

Обновить WriteProperties всем классам таким образом, чтобы в самом начале выводился тип объекта.

Создать лист объектов базового типа инициализированный как минимум 3-мя объектами разных типов. Вывести на экран их свойства WriteProperties().

Спасибо за внимание.