ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ ТЕХНОЛОГИЙ

2020г.

Основы ООП

- ✓ Принципы ООП
- ✓ Инкапсуляция
- ✓ Пространство имен
- ✓ Наследование классов
- ✓ Основы наследования
- ✓ Конструкторы и наследование
- ✓ Виртуальные методы
- ✓ Полиморфизм

Принципы ООП

Главное:
Классы – это их поведение и функциональность.
Инкапсулируйте все, что может изменяться.
Уделяйте больше внимания интерфейсам, а не их реализациям.
Каждый класс в вашем приложении должен иметь только одно назначение.

Базовые принципы ООП:
Абстракция – отделение концепции от ее экземпляра.

Наследование – способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода.
Наследственное отношение классов четко определяет их иерархию.

Полиморфизм – реализация задач одной и той же идеи разными способами.

Инкапсуляция – размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Абстракция

Абстракция – в основах ООП, это процесс скрытия всего, кроме релевантной информации об объекте.

Абстракция позволяет сосредоточиться на том, что делает объект, а не на том, как он это делает.

Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов.

Абстракция данных – популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования.

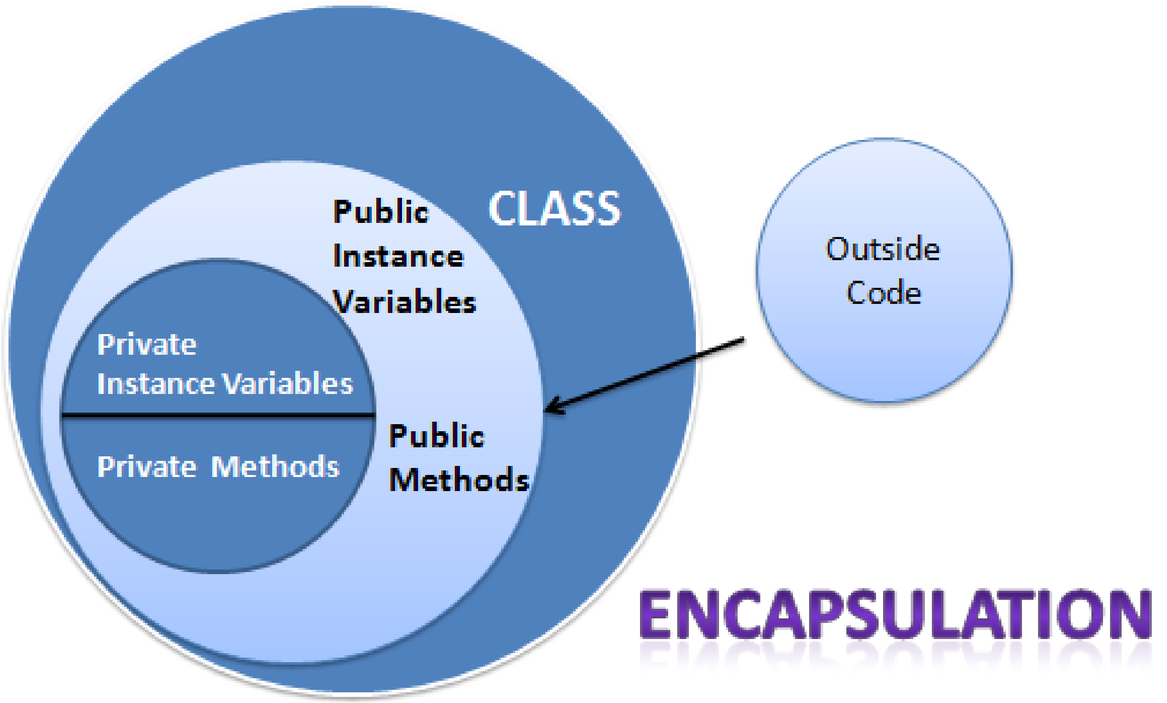
Абстракцию в ООП можно также определить, как способ представления элементов задачи из реального мира в виде объектов в программе. Абстракция всегда связана с обобщением некоторой информации о свойствах предметов или объектов, поэтому главное - это отделить значимую информацию от незначимой в контексте решаемой задачи. При этом уровней абстракции может быть несколько.

Инкапсуляция

Инкапсуляция – свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: **public**.

Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: **private**, **protected**, **internal**.



Пространство имен

Пространство имен (namespace) – концепция, позаимствованная из **C++** и позволяющая обеспечить уникальность всех имен, используемых в конкретной программе или проекте.

Логическое понятие, которое объединяет классы и типы, имеющие логические связи.

Иногда программисту при работе над крупным проектом не хватает удобочитаемых глобальных имен или нужны библиотеки классов сторонних разработчиков, в которых конфликтуют имена.

```
using System;
using alias = NamespaceName;
```

Наследование классов

Наследование – принцип ООП, согласно которому объект может наследовать атрибуты другого объекта.

В **C#** – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Класс, от которого произошло наследование, называется **базовым** или **родительским** (англ. **base class**). Классы, которые произошли от базового, называются **потомками**, **наследниками** или **производными** классами (англ. **derived class**).

Базовый класс может использоваться самостоятельно.

Базовый класс может использоваться для любого числа более конкретных классов.

В базовом классе определяются общие для объекта атрибуты.

Наследование классов

Класс, который наследует характеристики базового класс и имеет свои особенные характеристики – производный класс.

У производного класса может быть только один базовый класс. В **C#** нет множественного наследования.

Производный класс наследует все члены (поля, методы, свойства) базового класса.

В производном классе может быть добавлена своя собственная реализация.

В производном классе можно переопределить базовую реализацию - **override**.

Допускает создание иерархической классификации.

Использование “:” после имени класса.

Наследование классов

```
class Polygon {
    int height; int width; int
angleCount;
    Polygon(int h, int w, int
count)
    {
        height = h; width = w;
        angleCount = count;
    }

    public virtual double
GetSquare()  {
        return
0.5*(GetAp()*angleCount);
    }
}
```

```
class Rectangle : Polygon {
    Rectangle (int h, int w) :
base(h, w, 4) {}
    public override double
GetSquare()  {
        return height * width;
    }
}
class Square : Polygon
{
    Square (int h) : base(h, h, 4)
{}
    public override double
GetSquare()  {
        return height * height;
    }
}
```

Другие отношения между классами

В объектно-ориентированных языках программирования существует три способа организации взаимодействия между классами:

Наследование – описанно выше.

Ассоциация – это когда один класс включает в себя другой класс в качестве одного из полей.

Выделяют два частных случая ассоциации:

- **Композиция** – включение объектом-контейнером объекта-содержимого и управление его поведением; последний не может существовать вне первого.
- **Агрегация** – включение объектом-контейнером ссылки на объект-содержимое, при уничтожении первого последний продолжает существование.

Делегация – перепоручение задачи от внешнего объекта внутреннему.

Конструкторы и наследование

Конструкторы не передаются производному классу при наследовании. И если в базовом классе не определен конструктор по умолчанию без параметров, а только конструкторы с параметрами, то в производном классе мы обязательно должны вызвать один из этих конструкторов через ключевое слово **base**.

При вызове конструктора класса сначала отрабатывают конструкторы базовых классов и только затем конструкторы производных.

Конструктор базового класса отсутствует, конструктор производного класса присутствует:

- В базовом классе вызывается конструктор по умолчанию.

Конструктор базового класса присутствует, конструктор производного класса отсутствует.

- Конструктор базового класса параметризированный: Ошибка компилятора – нет возможности задать параметры конструктора базового класса.
- Конструктор базового класса переопределенный по умолчанию – все в порядке.

Виртуальные методы

При наследовании нередко возникает необходимость изменить в классе-наследнике функционал метода, который был унаследован от базового класса. В этом случае класс-наследник может переопределять методы и свойства базового класса.

Такие методы и свойства, в базовом классе помечаются модификатором **virtual и** называются виртуальными.

А чтобы переопределить метод в классе-наследнике, этот метод определяется с модификатором **override**. Переопределенный метод в классе-наследнике должен иметь тот же набор параметров, что и виртуальный метод в базовом классе.

Ключевое слово **virtual** – указывает на возможность переопределения метода.

Выбирается тот вариант метода, который следует вызвать, исходя из типа объекта к которому происходит обращение по ссылке.

Виртуальный метод не может быть объявлен как **static** или **abstract**.

Сокрытие метода базового класса

Сокрытие представляет определение в классе-наследнике метода или свойства, которые соответствует по имени и набору параметров методу или свойству базового класса. Для сокрытия членов класса применяется ключевое слово **new**.

Для того, чтобы обратиться к реализации свойства или метода в базовом классе, необходимо использовать ключевое слово **base** и через него обращаться к функциональности базового класса.

Сокрытие можно применять к переменным и константам, также используя ключевое слово **new.**

```
   public virtual string MyMethod()  {
      return “1”;
   } // метод базового класса

   public override string MyMethod() {
      return “2”;
   } // переопределенный метод
```

```
public new string MyMethod() {
      return “3”;
} //метод скрывающий базовую реализацию
```

Полиморфизм

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом(базовым типом) без информации о конкретном типе и внутренней структуре объекта.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию - например, реализация класса может быть изменена в процессе наследования.

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному - интерфейс, класс. Общность имеет внешнее и внутреннее выражение:

Реализация полиморфизма выполняется за счет использования:

- Переопределения методов (**virtual**, **override**).
- Интерфейсов.

SOLID - принципы

SOLID-принципы, это пять основных принципов объектно-ориентированного программирования и проектирования.

Принцип единственной ответственности (The Single Responsibility Principle)

Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его работы, должны быть инкапсулированы в этот класс и подчинены этой задаче.

Принцип открытости/закрытости (The Open Closed Principle)

Программные сущности должны быть открыты для расширения, но закрыты для изменений.

Принцип подстановки Барбары Лисков (The Liskov Substitution Principle)

Методы, использующие некий тип, должны иметь возможность использовать его подтипы, не зная об этом.

SOLID - принципы

Принцип разделения интерфейса (The Interface Segregation Principle)

Предпочтительнее разделять интерфейсы на более мелкие тематические, чтобы реализующие их классы не были вынуждены определять методы, которые в них не используются.

Принцип инверсии зависимостей (The Dependency Inversion Principle)

Система должна конструироваться на основе абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Дополнительно: Не повторяйся (Don’t repeat yourself – DRY)

Избегайте повторного написания кода, вынося в абстракции часто используемые задачи и данные. Каждая часть вашего кода или информации должна находиться в единственном числе в единственном доступном месте.

Задание

Создайте базовый класс описывающий транспорт с произвольными полями и методом **Move()** возвращающим способ передвижения в виде строки.

Добавьте производный класс описывающий лодку. Переопределите метод **Move()** чтобы его поведение было актуально.

Добавьте производный класс описывающий автомобиль. Переопределите метод **Move()** чтобы его поведение было актуально.

Добавьте производный класс описывающий Человека. Добавьте скрытое поле типа Транспорт. Создайте метод выводящий информацию о том, как Человек передвигается. Смена типа транспорта возможна исключитеьно через свойство либо метод.

В методе **Main()** класса **Program** объявите переменную типа транспорт и присвойте экземпляр любого производного класса.

Проверьте поведение вашего кода для различных экземпляров производных классов.