**Лекция 1. Принципы и основные понятия ооп**

**Объект** – сущность в виртуальном пространстве, обладающая определенным состоянием и поведением, имеющая заданные значения свойств и операций над ними

При создании объектно-ориентированной программы предметная область представляется в виде совокупности объектов. Выполнение программы состоит в том, что объекты обмениваются сообщениями.

Выделение существенных с той или иной точки зрения называется абстрагированием. Программный объект – абстракция. (выделяем только те свойства, которые нам нужны для программы)

Внутренние структуры данных и их алгоритмы обработки скрыты. Детали скрыты во избежание непреднамеренных изменений. Объект используется через его интерфейс – совокупность правил доступа.

**Инкапсуляция** – принцип ооп, суть в сокрытии деталей реализации. Она позволяет изменить реализацию объекта без модификации основной части программы, если его интерфейс остался прежним.

Преимущество ооп состоит в том, что для объекта можно определить наследников, корректирующих или дополняющих их поведение. При этом повторять код родительского объекта не нужно

**Наследование** – принцип ооп, позволяющий создавать новые классы на основе существующих.

Используется:

* Исключение из программы повторяющихся фрагментов кода
* Упрощения модификации программы
* Упрощения создания новых программ на основе существующих

**Полиморфизм** – принцип ооп, под которым понимается возможность во время выполнения программы использования с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа

**Класс** – обобщенное понятие, определяющее характеристики и поведение некоторого множества объектов, называемых экземплярами класса. В программном понимании **класс** является типом данных, определяемых пользователем, в котором объединены структуры данных и методы их обработки

[атрибуты][спецификаторы] class имя\_класса [:предки]

{

тело\_класса

}

Таблица 1. Значение спецификаторов

|  |  |
| --- | --- |
| abstract | Абстрактный класс. Применяется в иерархии объектов |
| internal | Доступ только из данного проекта (сборки) |
| new | Задает новое описание класса взамен унаследованного от предка. Используется для вложения классов (в иерархии объектов) |
| private | Доступ только из элементов класса, внутри которых описан данный класс. Используется для вложенных классов |
| protected | Доступ только из данного или производного класса. Используется для вложенных классов |
| protected internal | Доступ только из данного и производного класса в рамках текущего проекта (сборки) |
| public | Доступ к классу не ограничен |
| sealed | Бесплодный класс. Запрещает наследование данного класса. Применяется в иерархии объектов |
| static | Статический класс. Позволяет обращаться к методам класса без создания экземпляра класса |

В классе могут присутствовать статические члены, которые существуют в единственном экземпляре для всех объектов класса. Для работы с данными класса используются статические методы класса, для работы с данными экземпляра – методы экземпляра, или просто методы.

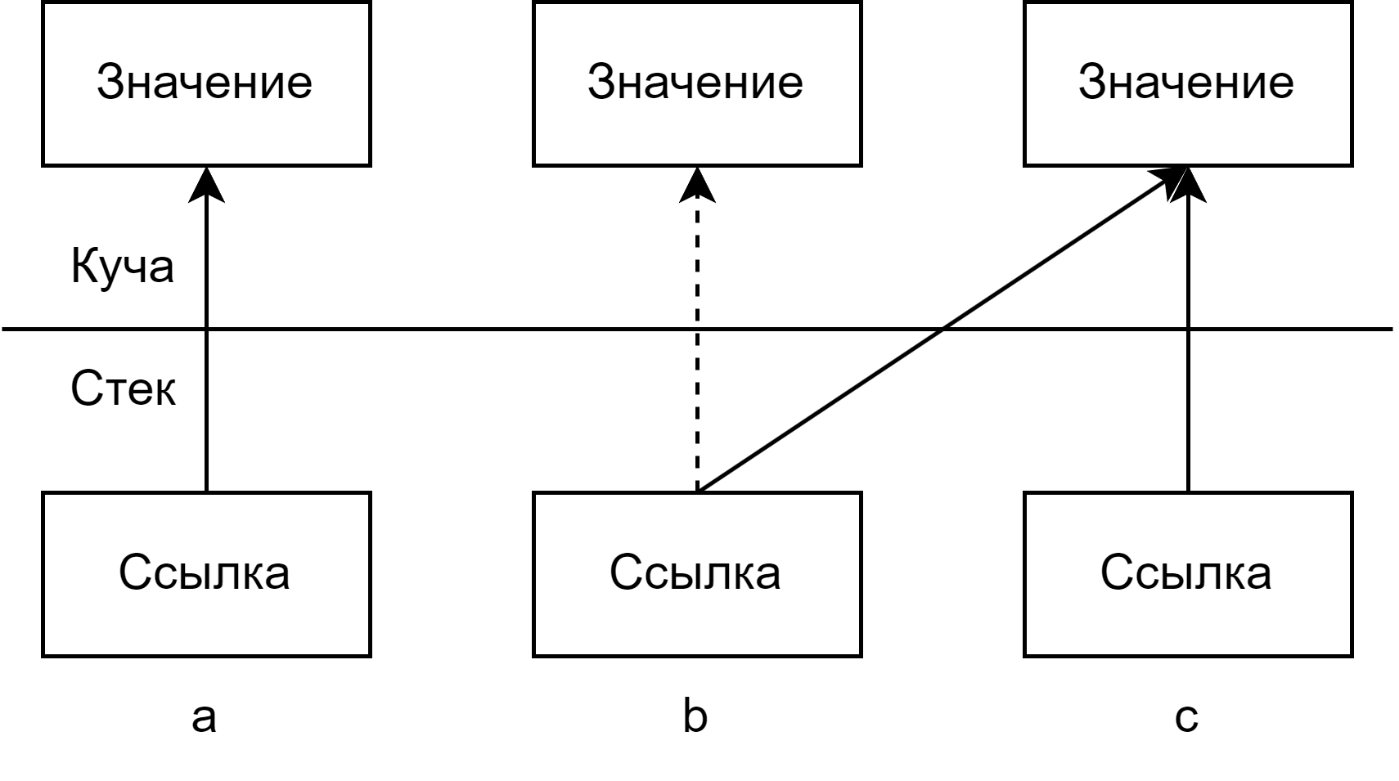
Класс может **содержать**:

* Данные, являющиеся переменными или константами
* Методы, реализующие не только вычисления, но и другие действия с классом или его экземпляром
* Конструкторы, реализующие действия по инициализации экземпляров класса или класса в целом
* Деструкторы, определяющие действия, которые необходимо выполнить непосредственно перед уничтожением объекта
* Свойства, определяющие возможности доступа к членам класса
* Индексаторы, обеспечивающие возможность доступа к членам класса по их порядковому номеру (индексу)
* Операции, задающие действия с экземплярами класса с помощью знаков операций
* События, определяющие уведомления, которые может генерировать класс
* Типы или структуры данных, определенные внутри класса

**Лекция 2.**

**Классы** относятся к ссылочным типам данных. Адрес переменной хранится в стеке, сам объект в динамической памяти. Данное различие существенно скажется на выполнении операций присваивания и сравнения объектов.

Имеются три объекта: a, b, c. Было выполнено следующее присваивание: b присвоить c. Теперь ссылка объекта b и c указывают на один и тот же объект. Старое значение b становится недоступным и удаляется сборщиком мусора.



**Сравнение.** Величины значимого типа равны, если равны их значения. Величины ссылочного типа равны, если они ссылаются на одну и ту же область памяти.

**Данные.**

Переменные могут быть константами и переменными.

[атрибуты][спецификатор][const] тип имя [= начальное\_значение];

Возможные спецификаторы для данных приведены в таблице 2.

Таблица 2. Спецификаторы данных.

|  |  |
| --- | --- |
| internal |  |
| new |  |
| private |  |
| protected |  |
| protected internal |  |
| public |  |
| readonly |  |
| static |  |
| volatile |  |

Пример класса:

class Circle

{

public int x = 0;

public int y = 0;

public int radius = 3;

public const double pi = 3.14;

public static readonly string name = "Окружность";

double area;

}

**Методы.**

Методы класса находятся в памяти в единственном экземпляре и используются всеми объектами одного класса совместно, поэтому необходимо обеспечить работу методов нестатических экземпляров с полями именного того объекта, для которого они были вызваны.

В явном виде параметр this применяется для того чтобы возвратить из методы ссылку на вызвавший объект, а также для идентификации поля в случае, если его имя совпадает с именем параметра метода, например:

public void Set(int x, int y, int radius)

{

// используем параметр this для обращения к полям класса, т.к. их имена совпадают с именами параметров метода

this.x = x;

thix.y = y;

this.radius = radius;

}

**Конструкторы.**

Метод класса, предназначенный для инициализации объекта. Вызывается автоматически при создании класса с помощью операции new. Имя конструктора совпадает с именем класса.

Имеет следующие свойства

1. Конструктор не возвращает значение, даже типа void.
2. Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации, выбор конструктора происходит автоматически, в зависимости от количества и типа передаваемых параметров
3. Если программист не указал ни одного конструктора или какие -то поля не были инициализированы, то полям значимых типов присваивается нуль, полям ссылочных типов – значение null.

public Circle(int radius) // конструктор 1

{

this.radius = radius;

}

public Circle(int x, int y, int radius) // конструктор 2

{

this.x = x;

thix.y = y;

this.radius = radius;

}

Можно еще такое извращение инициализатор. Вызовет конструктор 1 до конструктора 2

public Circle (int x, int y, int radius): this(radius)

{

this.x = x;

this.y = y;

}

**Деструкторы.**

Метод класса, который вызывается сборщиком мусора непосредственно перед удалением объекта из памяти. Деструктор обеспечивает корректность удаление объекта (закрытие файлов и т.д.)

[атрибуты][extern] ~имя\_класса()

{

тело деструктора

}

Свойства.

Метод класса. Открытые свойства связываются с закрытыми полями класса.

[атрибуты][спецификаторы] тип имя\_свойства

{

[get {код доступа для чтения}]

[set {код доступа для записи}]

}

Обращение к свойству:

x = имя\_класса.имя\_свойства;

имя\_класса.имя\_свойства = x;

Пример:

class Circle

{

public int X

{

get { return x; }

set { x = value; }

}

public int Y

{

get { return y; }

set { y = value; }

}

public int Radius

{

get { return radius; }

set { radius = value; }

}

private int x = 0;

private int y = 0;

private int radius = 3;

double area;

}