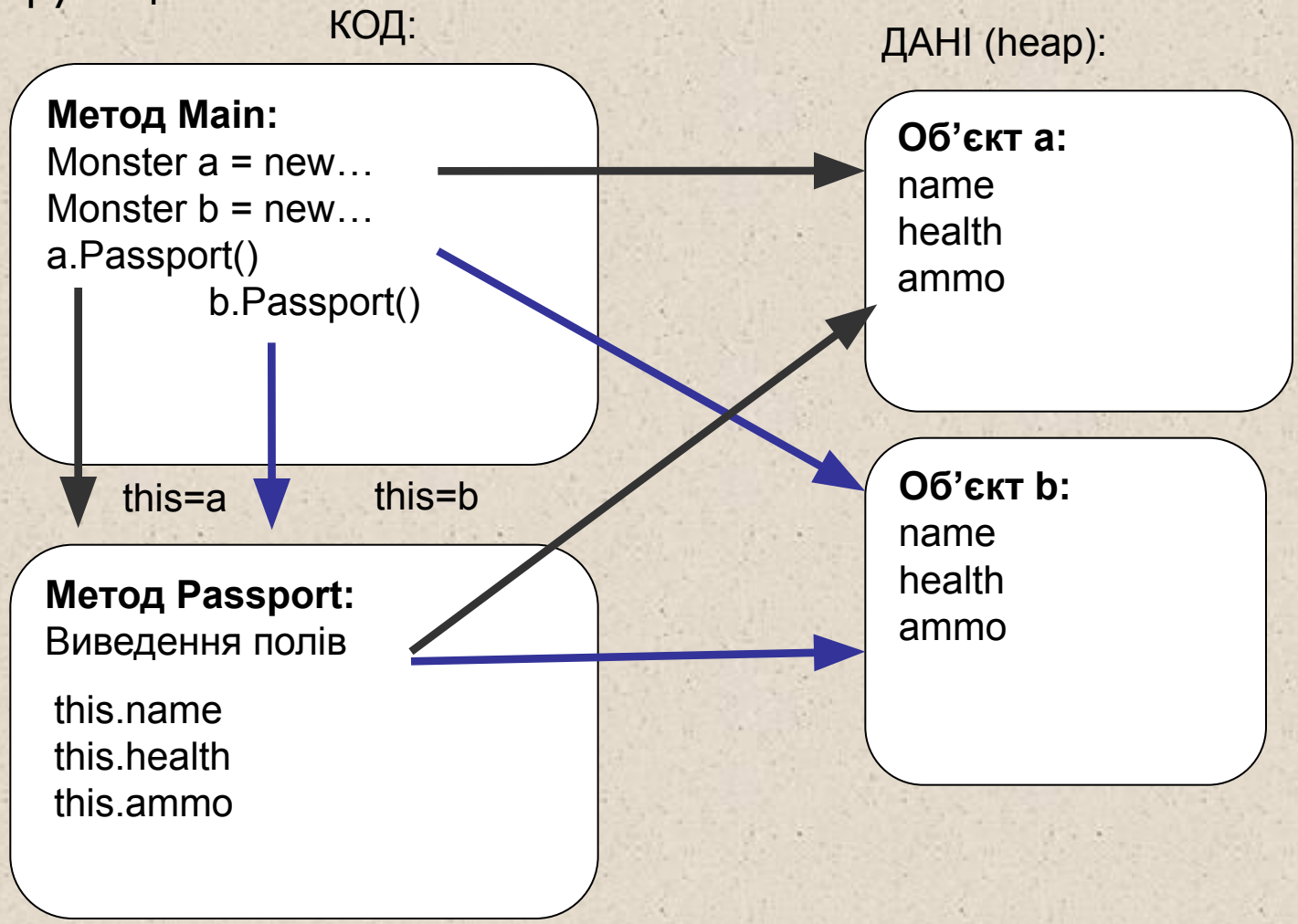
Ключове слово this

Щоб забезпечити роботу методу з полями того об’єкта, для якого він був викликаний, в метод автоматично передається прихований параметр this, в якому зберігається посилання на об’єкт, що викликав функцію.



Використання явного this

У явному вигляді параметр this застосовується:

1) щоб повернути з метода посилання на об’єкт, що його викликав:

```
class Demo
{       double y;
        public Demo T()     { return this; }

2) для ідентифікації поля, якщо його ім’я співпадає з іменем параметра метода:

        public void Set_y( double y ) { this.y = y; }
}
```

Приклад: лічильник (this)

```
class Counter
{     public bool Sync(out int x)
      {     x = n;
            return n==0 ? false : true ;
      }
      public void Set( int start ) { n = start; }
      int n;
       ...
}
class Program
{     static void Main(string[] args)
      {     Counter num1 = new Counter();
            Counter num2 = new Counter();
            ...
            int temp;
            if ( num1.Sync(out temp) ) num2.Set(temp);
            …
}}
```

```
public void Set( int n ) { this.n = n; }
```

Конструктори

Конструктор – особливий вид метода, що служить для ініціалізації

- **об’єкта** (конструктор екземпляра) або
- **класу** (статичний конструктор).

Конструктор екземпляра ініціалізує дані екземпляра, конструктор класу — дані класу.

Конструктори екземпляра

Конструктор викликається автоматично під час створення об’єкта класу за допомогою операції **new**. Ім’я конструктора співпадає з іменем класу.

Властивості конструкторів

- Конструктор не повертає значення, навіть типу void.
- Клас може мати декілька конструкторів з різними параметрами для різних видів ініціалізації.
- Якщо програміст не вказав ні одного конструктора або якісь поля не були проініціалізовані, то полям значимих типів присвоюється нуль, полям вказівникових типів — значення null.
- Конструктор, що викликається без параметрів, називається **конструктором за замовчуванням**.

Приклад класу з конструктором

```
class Demo
   {
      public Demo( int a, double y )        // конструктор
      {
         this.a = a;
         this.y = y;
      }
      int a;
      double y;
   }

   class Class1
   {   static void Main()
      {
         Demo a = new Demo( 300, 0.002 );     // виклик конструктора
         Demo b = new Demo( 1, 5.71 );        // виклик конструктора
         …
    } }
```

Приклад класу з двома конструкторами

```
class Demo
   {
      public Demo( int a )                  // конструктор 1
      {
         this.a = a;
         this.y = 0.002;
      }
      public Demo( double y )               // конструктор 2
      {
         this.a = 1;
         this.y = y;
      }
      ...
   }
      ...
         Demo x = new Demo( 300 );          // виклик конструктора 1
         Demo y = new Demo( 5.71 );         // виклик конструктора 2
```

Приклад: лічильник (конструктори)

```
class Counter
{
        public Counter( int n = 0)
        {
            this.n = n > 0 ? n : 0;
        }
        int n;
        ...
}
class Program
{     static void Main(string[] args)
        {    Counter num1 = new Counter();
             Counter num2 = new Counter(128);
             ...
        }
}
```

Приклад класу – 1/2

```
class Monster {
   public Monster()   // конст
   {
         name  = "Noname";
         health = 100;
         ammo = 100;
   }
  public Monster( string name ) : this()
   {
         this.name = name;
    }
  public Monster(string name, int health,
    int ammo)
    {
       this.name  = name;
       this.health = health > 0 ? health : 0 ;
       this.ammo = ammo > 0 ? ammo : 0 ;
    }
 public string GetName()     // метод
   {  return name;   }
 public int GetAmmo()       // метод
    {  return ammo;}
```

```
Monster Vasia = new Monster();
Monster Petya = new Monster("Петя");
Monster Masha = new Monster("Марія", 150, 3000);
```

```
public int Health {              // властивість
     {   Console.WriteLine(
  "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo );
      }

public override string ToString(){
     string buf = string.Format(
       "Monster {0} \t health = {1} \
     ammo = {2}", name, health, ammo);
    return buf;  }

  string name;
  int health, ammo;
}
```

Приклад класу – 2/2

```
class Monster {

 public Monster(string name = "Noname",
    int health = 100, int ammo = 100)
    {
        this.name  = name;
        this.health = health > 0 ? health : 0 ;
        this.ammo = ammo > 0 ? ammo : 0 ;
    }
 public string GetName()     // метод
   {  return name;   }
 public int GetAmmo()        // метод
    {  return ammo;}
```

```
public int Health {            // властивість
   get { return health;  }
   set { if (value > 0) health = value;
         else           health = 0;
       }
 }

 public void Passport()              // метод
     {   Console.WriteLine(
  "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo );
      }

public override string ToString(){
     string buf = string.Format(
      "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo);
    return buf;  }
```

```
Monster Vasia = new Monster();
Monster Petya = new Monster("Петя");
Monster Masha = new Monster("Марія", 150, 3000);
```

Статичні конструктори

- Статичний конструктор (конструктор класу) ініціалізує статичні поля класу.
- Він не має параметрів, його не можна викликати явним чином. Система сама визначає момент, під час якого потрібно його виконати. Гарантується, що це відбувається до створення першого екземпляра об'єкта і до виклику будь-якого статичного метода.
- Статичний конструктор, як і будь-який статичний метод, не має доступу до полів екземпляра.

Приклад класу зі статичними елементами

```
class Monster {
    public Monster(string name = "Noname", int health = 100, int ammo =
    100)
     {
         this.name  = name;
         this.health = health > 0
         this.ammo = ammo > 0
         ++num_Monsters;
     }
}
  static Monster() { num_Monsters = 0; }  // переобтяжено, для прикладу
  public static int Total() { return num_Monsters;  }
    ...
  string name; int health, ammo;
  static int num_Monsters = 0;
}
```

```
Monster Vasia = new Monster();
Monster Petya = new Monster("Петя");
Monster Masha = new Monster("Марія“,150, 300);
Console.WriteLine(Monster.Total());          // 3
```

Любой человек должен уметь менять пеленки, планировать вторжения, резать свиней, конструировать здания, управлять кораблями, писать сонеты, вести бухгалтерию, возводить стены, вправлять кости, облегчать смерть, исполнять приказы, отдавать приказы, сотрудничать, действовать самостоятельно, решать уравнения, анализировать новые проблемы, бросать навоз, ***программировать компьютеры****, вкусно готовить, хорошо сражаться, достойно умирать.*
Специализация — удел насекомых.

Властивості

- Властивості служать для організації доступу до полів класу. Як правило, властивість визначає методи доступу до закритого поля.
- Властивості забезпечують поділ між внутрішнім станом об'єкта і його інтерфейсом.
- Синтаксис властивості:

[специфікатори] тип ім'я_властивості

{

[get код_доступу]

[set код_доступу]

}

Під час звернення до властивості автоматично викликаються вказані в ній блоки читання (**get)** і установки (**set)**.

- Може бути відсутня або частина get, або set, але не обидві одночасно. Якщо відсутня частина set, властивість доступна тільки для читання (read-only), якщо відсутня get - тільки для запису (write-only).

Приклад: лічильник (властивості)

```
class Counter
{       public Counter( int n = 0 )
                { this.n = n > 0 ? n : 0; }
        public int N
        {  get { return n; }
           set { n = value > 0 ? value : 0; }
        }
// або: set { if (value > 0) n = value; else throw new Exception();}
        int n;        // поле, пов'язане з властивістю N
}
class Program
{     static void Main(string[] args)
      {    Counter num = new Counter();
           num.N = 5;          //   працює set
           int a = num.N;      //   працює get
           num.N++;            //   працює get, а потім set
           ++num.N;            //   працює get, а потім set
}}
```

Ще приклад опису властивостей

```
public class Button: Control
{  private string caption;          // поле, з яким пов'язана властивість
   public string Caption {          // властивість
      get { return caption; }       // спосіб отримати властивість

      set                           // спосіб установки властивості
         { if (caption != value) { caption = value; }
}}   ...
```

У програмі властивість виглядає як поле класу:

```
Button ok = new Button();
ok.Caption = "OK";          // викликається метод установки властивості
string s = ok.Caption;      // викликається метод отримання властивості
```

Приклад класу: властивості

```
class Monster {
    public Monster(string name = "Noname",
     int health = 100, int ammo = 100)
    {
        this.name  = name;
        this.health = health > 0 ? health : 0 ;
        this.ammo = ammo > 0 ? ammo : 0 ;
    }

  public string GetName()     // метод
    {  return name;   }
  public int GetAmmo()        // метод
    {  return ammo;}
```

```
public int Health {      // властивість
   get { return health;  }
   set {health = value > 0 ? value : 0;
         }
 }
public string Name {  // властивість
    get { return name; }
}
 public void Passport()              // метод
     {   Console.WriteLine(
  "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo );
      }
public override string ToString(){
     string buf = string.Format(
       "Monster {0} \t health = {1} \
     ammo = {2}", name, health, ammo);
    return buf;  }

  string name;
  int health, ammo;
}
```

Приклад властивостей, що обраховуються

```
namespace ConsoleApplication1
{   class Ball {
      public Ball(double radius, double density)
      {  this.radius = radius;
         this.density = density;
      }
      public double Mass
      {   get { return radius * density; } }
      public double Diameter
      {   get { return radius * 2; }   }

      double radius;
      double density;
   }
```

```
class Program
   {  static void Main(string[] args)
      {
         Ball ball = new Ball(10, 6.2);
         Console.WriteLine("Діаметр: " +
                                ball.Diameter +
                        " Маса: " + ball.Mass);
} } }
```

Перегрузка методів

- *Перегрузкою методів* називається використання декількох методів з одним і тим же іменем, але різними типами параметрів.
- Компілятор визначає, який саме метод потрібно викликати, за типом фактичних параметрів. Це називається ***дозволом*** (resolution) перегрузки.

```
                    // Повертає найбільше з двох цілих:
int max( int a, int b )
                   // Повертає найбільше з трьох цілих:
int max( int a, int b, int c )
                  // Повертає найбільше з першого параметра
                  // і довжини другого:
int max ( int a, string b )
...
Console.WriteLine( max( 1, 2 ) );
Console.WriteLine( max( 1, 2, 3 ) );
Console.WriteLine( max( 1, "2" ) );
```

- Перегрузка методів є проявом *поліморфізму*

Приклад: лічильник (перегрузка)

```
class Counter
{       public Counter() { }
        public Counter( int n ) { this.n = n > 0 ? n : 0;  }

        public void Inc() { ++n; }
        public void Inc( int delta ){ n += Math.Abs(delta); }
        int n;
        ...
}
class Program
{     static void Main(string[] args)
      {     Counter num1 = new Counter();
            Counter num2 = new Counter(128);
            num1.Inc(4);
            num1.Inc();
            ...
      }
}
```

Перевантажені конструктори

Перевантажені методи

Приклад класу

```
class Monster {
   public Monster()   // конструктор
   {
         this.name  = "Noname";
         this.health = 100;
         this.ammo = 100;
   }
  public Monster( string name ) : this()
   {
         this.name = name;
    }
  public Monster( int health, int ammo,
    string name )
    {
         this.name  = name;
         this.health = health;
         this.ammo = ammo;
    }
  public string GetName()     // метод
    {  return name;   }
  public int GetAmmo()       // метод
    {  return ammo;}
```

```
public int Health {              // властивість
   get { return health;  }
   set { if (value > 0) health = value;
          else              health = 0;
         }
 }

 public void Passport()              // метод
     {   Console.WriteLine(
  "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo );
      }

public override string ToString(){
     string buf = string.Format(
       "Monster {0} \t health = {1} \
     ammo = {2}", name, health, ammo);
    return buf;  }

  string name;
  int health, ammo;
}
```