Обектно ориентирано програмиране - курс

- Основни принципи на дизайн
- Windows Forms (контроли, рисуване)
- LINQ
- Сериализация
- Библиотеки
- Reflection

Обектно ориентирано програмиране

- Обектно ориентираното програмиране е подход, който залага на идеята, че всяка програма работи с данни, описващи предмети и явления от реалния живот.

Основни понятия

- **Обект -** елементарна единица в ООП, включваща в себе си както описващите обекта данни, така и средствата за обработка на тези данни;

- **Клас -** обобщено описание на набор обекти, притежаващи някои еднакви методи и структури от данни.

Основни концепции в ООП

- **Капсулация -** обединяване на данните с процедурите и функциите в рамките на единно цяло;
- **Наследяване -** възможност за създаване на йерархия на обектите чрез наследяване на техните характеристики;
- **Полиморфизъм -** възможност за третирането на обекти от наследен клас като обекти от негов базов клас.

Структура от данни

- Потребителски тип, съдържащ полета (член-променливи) и функции (методи)

Структура от данни

- Последователна памет*
- Всяко поле може да бъде прочетено и записано
- Полетата на структура могат да бъдат достъпени и всяко може да бъде третирано като стандартна променлива

* Размера може да е по-голям от сумата на съставящите го стойности

Структура

- [<модификатори>] struct <идентификатор> {
  [<декларация на член1>
  <декларация на член2>
  …
  <декларация на членN>]
  }
- **декларация на член**: <декларация на конструктор> | <декларация на член-променлива> | <декларация на член-функция> | <декларация на свойство>

Декларация на структура

- **декларация на член-променлива**:
  [<модификатори>] <тип> <идентификатор> [=<стойност>];
- **декларация на член-функция**:
  <декларация на функция>
- **декларация на конструктор**:
  [<модификатори>]
  <идентификатор на структурата>(<аргумент1>[,<аргументN>]) {
  <оператори>
  }
- **модификатори**: private | public | internal | <други модификатори>

Модификатори за достъп

- private – видим в текущата структура
- internal – видим в същото асембли
- public – видим от всякъде

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;
}

static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle();

    rectangle.x = 10;
    rectangle.y = 15;

    int surface = rectangle.x * rectangle.y;
}
```

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;
}
static int RectangleSurface(Rectangle rectangle)
{
    return rectangle.x * rectangle.y;
}
static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle();

    rectangle.x = 10;
    rectangle.y = 15;

    int surface = RectangleSurface(rectangle);
}
```

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;

    public static int RectangleSurface(Rectangle rectangle)
    {
        return rectangle.x * rectangle.y;
    }
}

static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle();

    rectangle.x = 10;
    rectangle.y = 15;

    int surface = Rectangle.RectangleSurface(rectangle);
}
```

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;

    public int RectangleSurface()
    {
        return x * y;
    }
}

static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle();

    rectangle.x = 10;
    rectangle.y = 15;

    int surface = rectangle.RectangleSurface();
}
```

this

- Това е неявно съществуваща променлива в тялото на член-функции, които **не са** маркирани като **static**

- **this** е обектът, за който е извикана член-функцията

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;

    public int RectangleSurface()
    {
        return this.x * this.y;
    }
}
static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle();

    rectangle.x = 10;
    rectangle.y = 15;

    int surface = rectangle.RectangleSurface();
}
```

Структури от данни

```
public struct Rectangle
{
    public int x;
    public int y;

    public Rectangle(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int RectangleSurface()
    {
        return x * y;
    }
}

static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle(10, 15);

    int surface = rectangle.RectangleSurface();
}
```

Задача

- Да се създаде тип за триъгълник
- Триъгълника да има височина и основа към нея
- Да се имплементира метод за изчисляване на лице
- Да се създаде масив от три триъгълника и да се въведат от клавиатурата съответните височини и основи
- Да се изведат лицата на триъгълниците

Задача

```
public struct Triangle
{
    public int h;
    public int b;

    public Triangle(int h, int b)
    {
        this.h = h;
        this.b = b;
    }

    public int Surface()
    {
        return h * b / 2;
    }
}
```

```
static void Main(string[] args)
{
    Triangle[] triangles = new Triangle[3];

    for(int i= 0; i< triangles.Length; i++)
    {
        Console.Write("Triangle " + (i + 1) + " b:");
        triangles[i].b = int.Parse(Console.ReadLine());
        Console.Write("Triangle " + (i + 1) + " h:");
        triangles[i].h = int.Parse(Console.ReadLine());
    }

    for(int i= 0; i< triangles.Length; i++)
    {
        Console.WriteLine(
            "Triangle '" + (i + 1) + "' surface area:" + triangles[i].Surface());
    }
}
```

Класове

- Потребителски тип, съдържащ полета (член-променливи) и функции (методи)

- Променливите от този тип могат да бъдат **празни** (стойност null)

Декларация на клас

- [<модификатори>] class <идентификатор>[: <базови типове>] {
  [<декларация на член1>
  <декларация на член2>
  …
  <декларация на членN>]
  }
- **декларация на член**: <декларация на конструктор> | <декларация на член-променлива> | <декларация на член-функция> | <декларация на свойство> | <декларация на деструктор>

Декларация на клас

- **декларация на деструктор**:
  [<модификатори>]
  ~<идентификатор на класа>() {
  <оператори>
  }
- **базови типове**:
  <базов тип1>[, <базов тип2>[,… <базов типN>]]
- **базов тип**:
  <клас>|<интерфейс>
  - може да има само един базов клас, но множество интерфейси; ако не е зададен базов клас, базовият клас е object

Класове

```
public class Rectangle
{
    public int x;
    public int y;

    public Rectangle(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int RectangleSurface()
    {
        return x * y;
    }
}

static void Main(string[] args)
{
    Rectangle rectangle = new Rectangle(10, 15);

    int surface = rectangle.RectangleSurface();
}
```

Задача

```
public struct Triangle
{      class
    public int h;
    public int b;


    public Triangle(int h, int b)
    {
        this.h = h;
        this.b = b;
    }


    public int Surface()
    {
        return h * b / 2;
    }
}
```

```
static void Main(string[] args)
{
    Triangle[] triangles = new Triangle[3];

    for(int i= 0; i< triangles.Length; i++)
    {
        Console.Write("Triangle " + (i + 1) + " b:");
        triangles[i].b = int.Parse(Console.ReadLine());
        Console.Write("Triangle " + (i + 1) + " h:");
        triangles[i].h = int.Parse(Console.ReadLine());
    }

    for(int i= 0; i< triangles.Length; i++)
    {
        Console.WriteLine(
            "Triangle '" + (i + 1) + "' surface area:" + triangles[i].Surface());
    }
}
```

```
triangles[i] = new Triangle(0, 0);
```

Типът Object

```
public class Object
{
        public Object();

        public virtual bool Equals(object obj);
        public static bool Equals(object objA, object objB);
        public virtual int GetHashCode();
        public Type GetType();
        protected object MemberwiseClone();
        public static bool ReferenceEquals(object objA, object objB);
        public virtual string ToString();
}
```

Променлива VS Инстанция

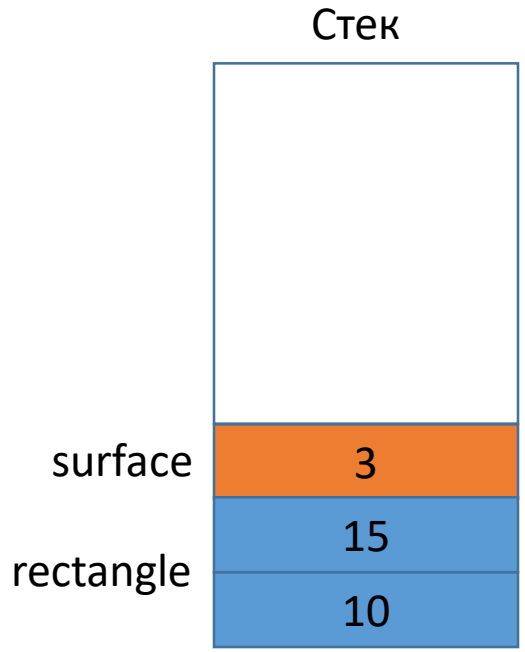
- **Инстанция** – представлява памет заделена и инициализирана съгласно някакъв тип

- **Променлива** – основна езикова единица, чрез която се извършва достъп до инстанция от определен тип

Променлива VS Инстанция

```
Rectangle rectangle = new Rectangle(10, 15);

int surface = rectangle.RectangleSurface();
```



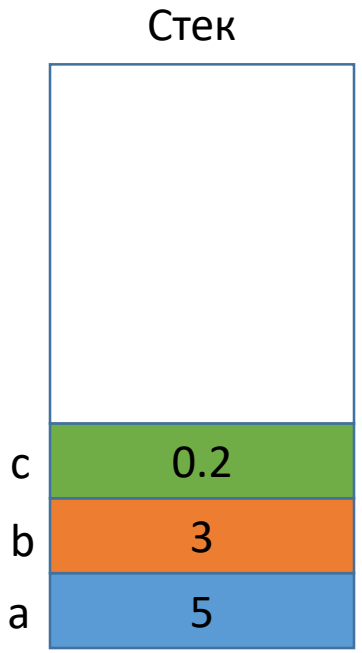
Стойностни типове (value types)

- Определение: Типове разполагани в стека за изпълнение на програмата. Инстанциите от този тип представят директно стойностите съхранени в тях.

Стойностни типове (value types)

```
static void Main(string[] args)
{
    int a = 5;
    int b = 3;
    double c = 0.2;

    //...
}
```



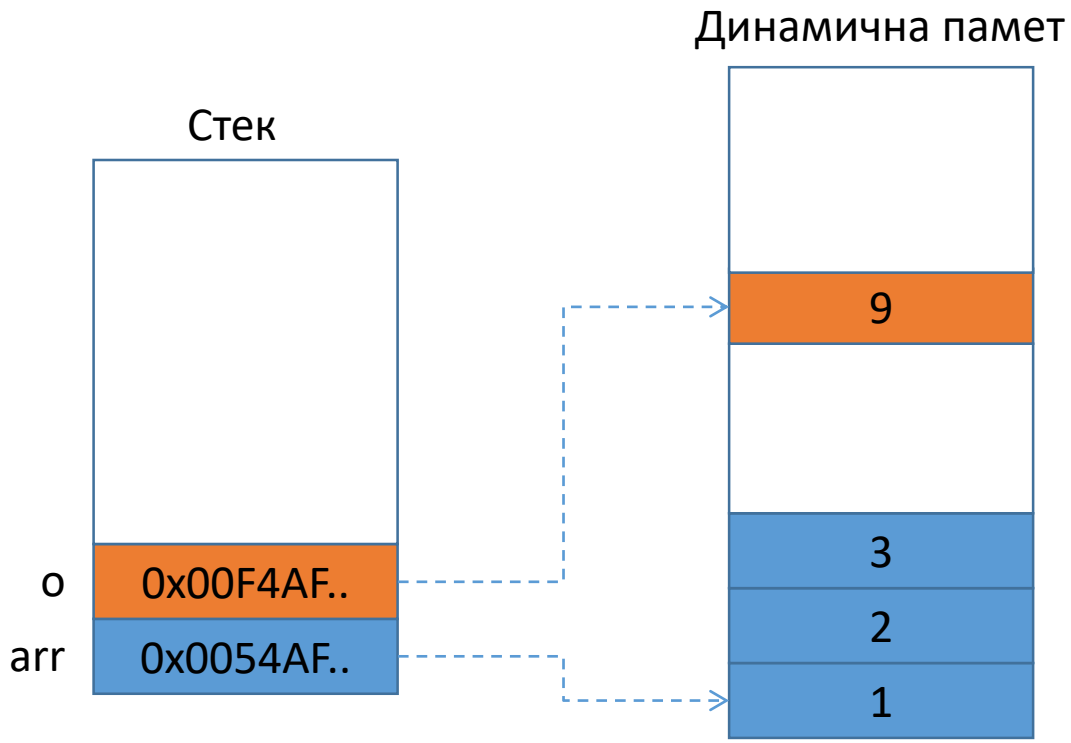
Референтни типове (reference types)

- Определение: Типове разполагани в динамичната памет (heap). Инстанциите от този тип са указател към паметта, в която е разположена стойността.

Референтни типове (reference types)

```
static void Main(string[] args)
{
    int[] arr = new int[] {1,2,3};
    object o = 9;

    //...
}
```



Стойностни и референтни типове

- Структурите са стойностни типове
- Класовете са референтни типове

Параметри подавани по стойност

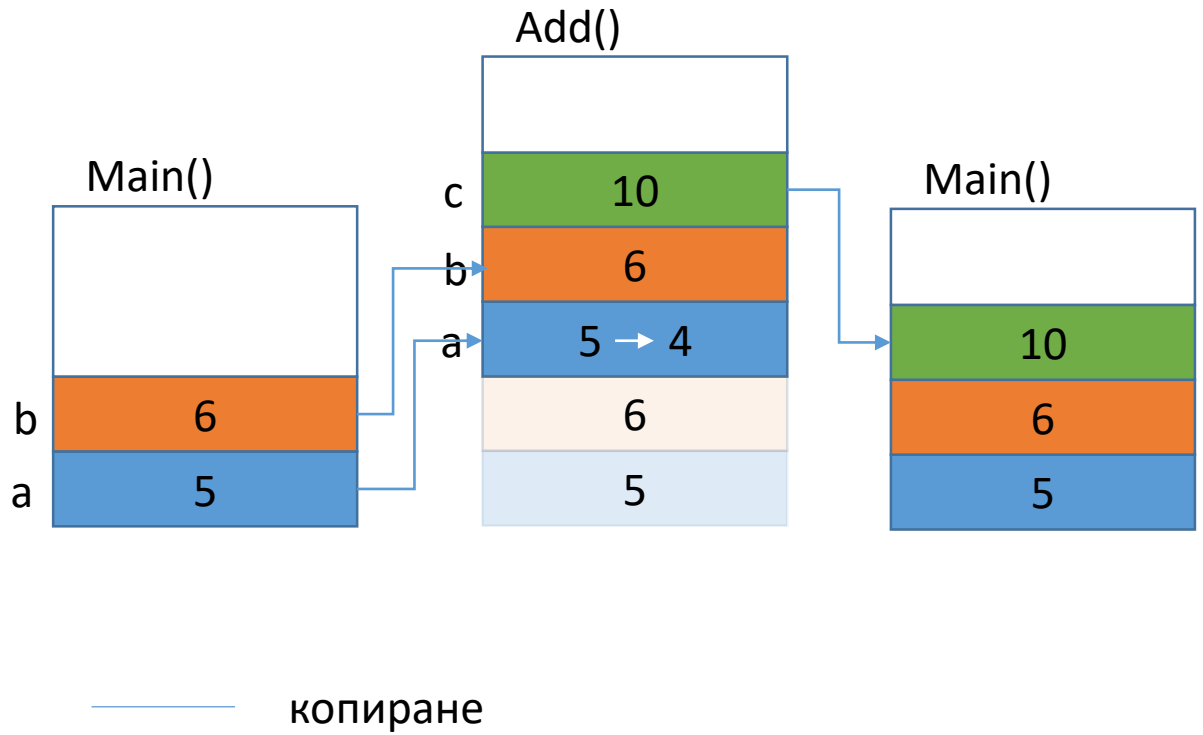
```
public static int Add(int a, int b)
{
    a = 4;
    return a + b;
}

static void Main(string[] args)
{
    int a = 5, b = 6;
    int res = Add(a, b);
    // a = ?
}
```

Параметри подавани по стойност

```
public static int Add(int a, int b)
{
    a = 4;
    return a + b;
}

static void Main(string[] args)
{
    int a = 5, b = 6;
    int res = Add(a, b);
    // a = ?
}
```



Параметри подавани по референция

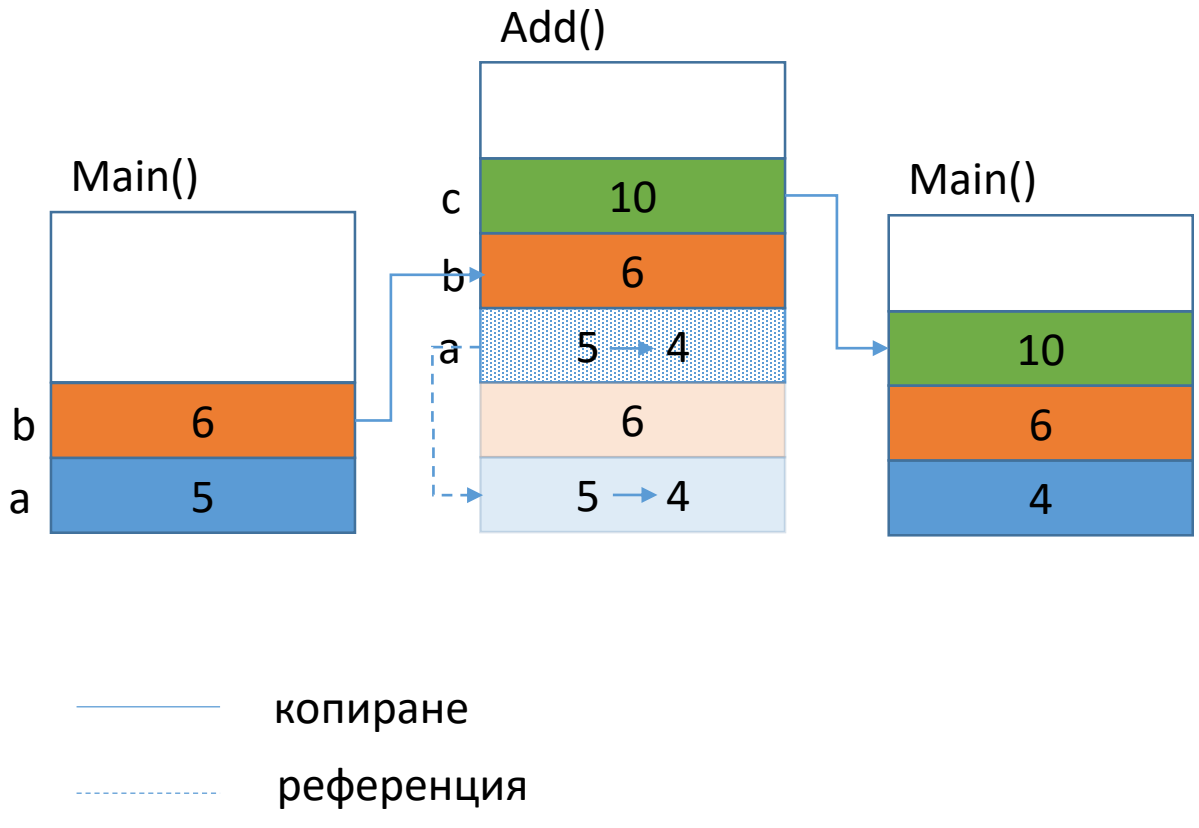
```
public static int Add(ref int a, int b)
{
    a = 4;
    return a + b;
}

static void Main(string[] args)
{
    int a = 5, b = 6;
    int res = Add(ref a, b);
    // a = ?
}
```

Параметри подавани по референция

```
public static int Add(ref int a, int b)
{
    a = 4;
    return a + b;
}

static void Main(string[] args)
{
    int a = 5, b = 6;
    int res = Add(ref a, b);
    // a = ?
}
```

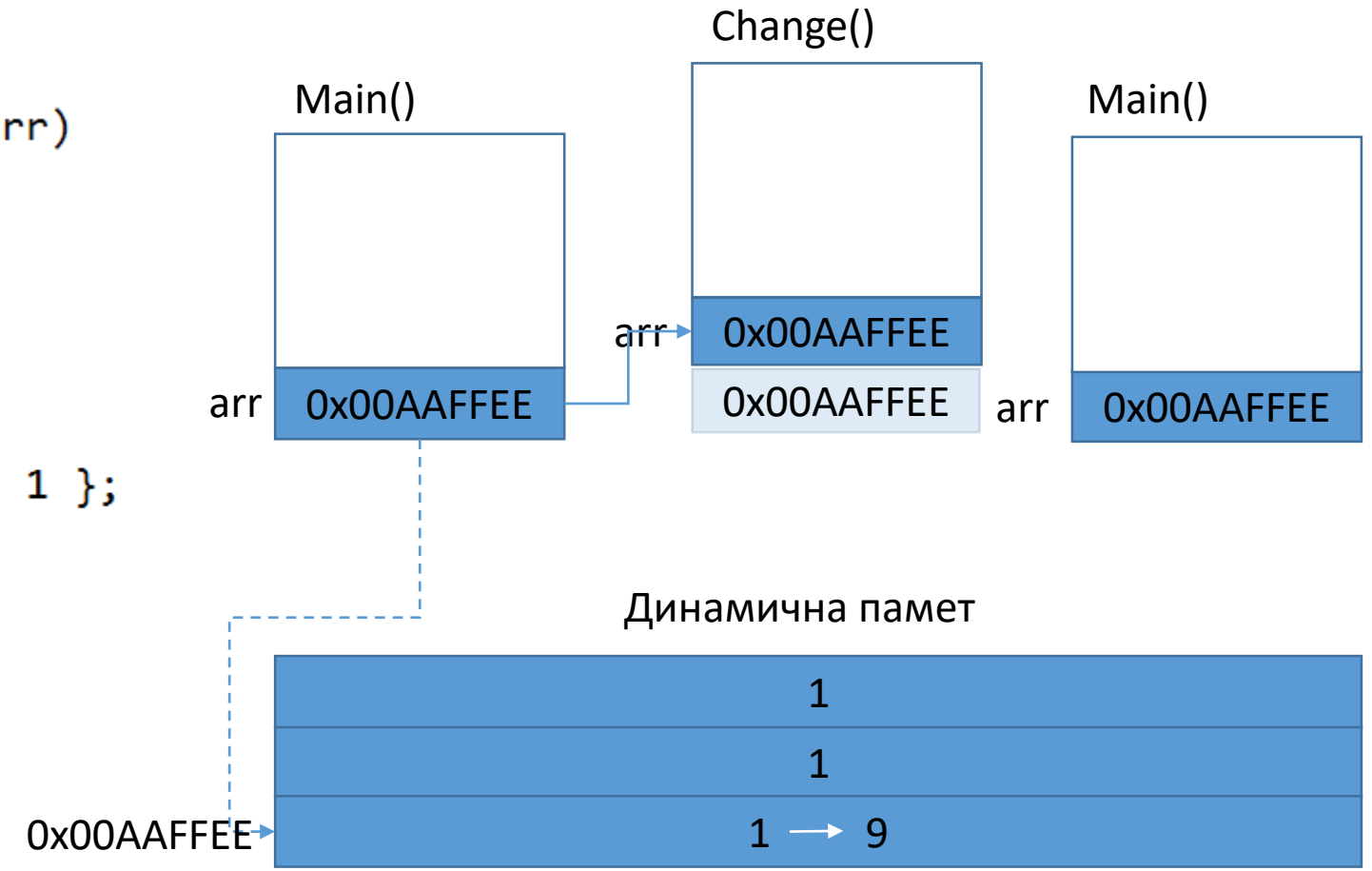


Параметри от референтен тип

```
public static void Change(int[] arr)
{
    arr[0] = 9;
}
static void Main(string[] args)
{
    int[] arr = new int[] { 1, 1, 1 };
    Change(arr);
    // arr[0] = ?
}
```

Параметри от референтен тип

```
public static void Change(int[] arr)
{
    arr[0] = 9;
}
static void Main(string[] args)
{
    int[] arr = new int[] { 1, 1, 1 };
    Change(arr);
    // arr[0] = ?
}
```

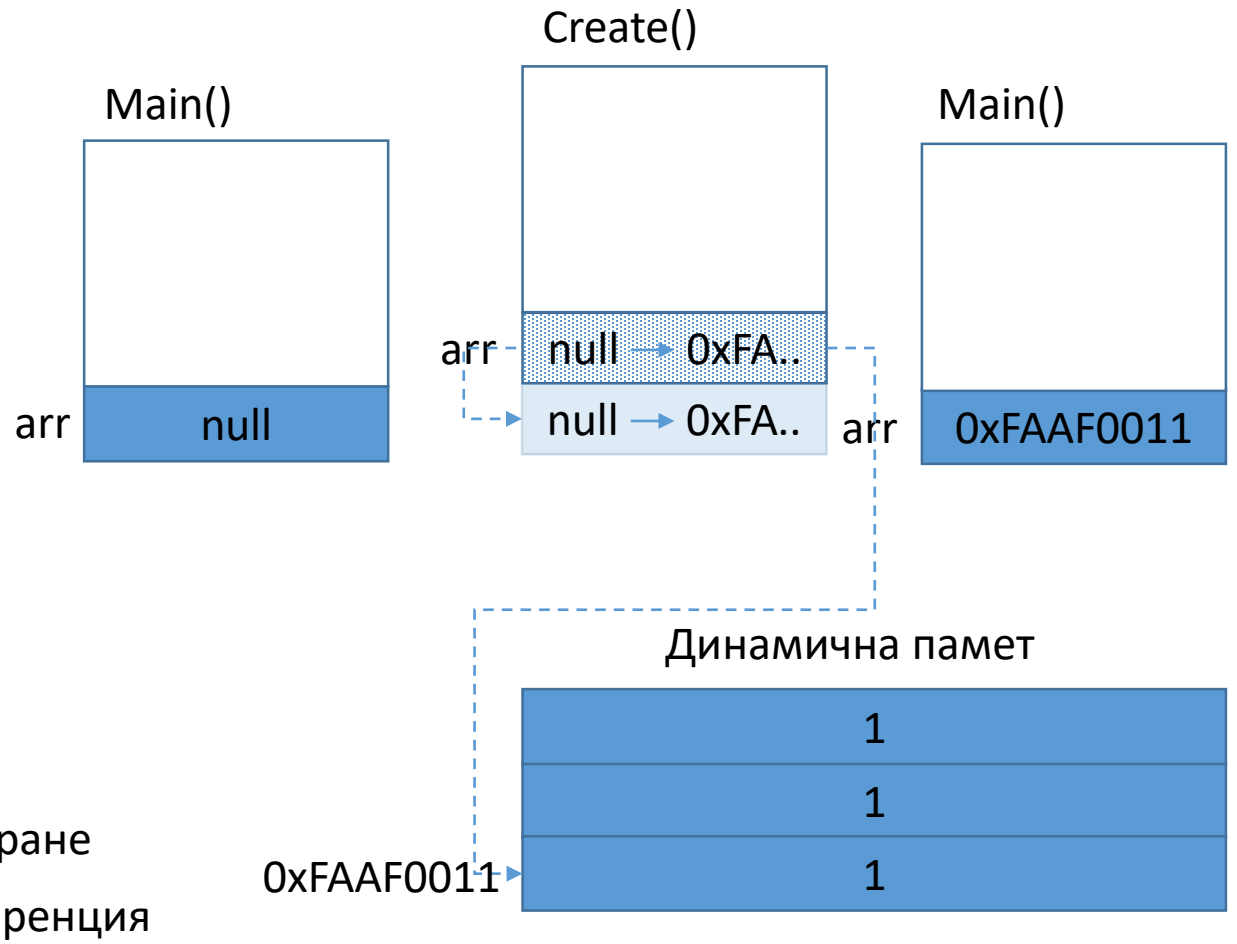


Параметри от референтен тип подавани по референция

```
public static void Create(out int[] arr)
{
    arr = new int[]{1, 1, 1};
}
static void Main(string[] args)
{
    int[] arr;
    Create(out arr);
}
```

Параметри от референтен тип подавани по референция

```
public static void Create(out int[] arr)
{
    arr = new int[]{1, 1, 1};
}
static void Main(string[] args)
{
    int[] arr;
    Create(out arr);
}
```



Програма телефонен указател

- Всеки запис за лице съдържа име, тел. номер и имейл (потребителски тип данни)
- Записите се съхраняват в динамичен масив
- Указателя реализира функции по добавяне, търсене и изтриване на телефон от масива (с избор от конзолата)
- * При изключване на програмата масива се записва във файл, а при стартиране се инициализира с данните от файла

Програма телефонен указател 2

- Телефонния указател да се капсулира в отделен потребителски тип
- Да се разширят функциите на програмата, така че всяко лице да има произволен брой номера и имейл адреси
- Да се реализира функция по редакция на съществуващ запис