

# Използване на променливи, изрази и константи

Правилна организация  
на изрази и данни



Учителски екип

Обучение за ИТ кариера

<https://it-kariera.mon.bg/e-learning/>



# Съдържание

- Принципи за инициализация
- Обхват, живот, времетраене
- Използване на променливи
  - Именуване на променливи
  - Конвенции при именуването
  - Стандартни представки
- Използване на изрази
- Използване на константи



# Инициализация на променливи

- Инициализирайте всички променливи преди да ги ползвате
  - Локалните променливи да се инициализират ръчно
  - Декларирайте и дефинирайте всяка променлива близо до мястото, където се използва
  - Този C# код ще доведе до грешка в компилирането:

```
int value;  
Console.WriteLine(value);
```



- Може да инициализираме променливата при декларирането:

```
int value = 0;  
Console.WriteLine(value);
```



# Инициализация на променливи (2)

- Особено внимавайте за **броячи и колектори**
  - Честа грешка е да забравите да нулирате брояч или колектор

```
int sum = 0;
for (int i = 0; i < array.GetLength(0); i++)
{
    for (int j = 0; j < array.GetLength(1); j++)
    {
        sum = sum + array[i, j];
    }

    Console.WriteLine(
        "The sum of the elements in row {0} is {1}", i, sum);
}
```



Сумата трябва да се нулира след края на вътрешния цикъл for



# Инициализация на променливи (3)

- Вижте необходима ли е **повторна инициализация**
  - Уверете се, че изразът за инициализация е в тази част от кода, която се повтаря
- Проверете **валидността** на входните **параметри**
  - Преди да присвоявате каквито и да е входни данни от конзолата, уверете се, че стойностите са адекватни

```
int input;  
bool validInput = int.TryParse(Console.ReadLine(), out input);  
if (validInput)  
{  
    ...  
}
```



# Отчасти инициализирани обекти

- Уверете се, че обектите **не могат** да са **инициализирани само отчасти**
  - Направете всички полета **private** и изисквайте валидни стойности за всички задължителни полета в конструкторите
  - Пример: обектът **Student** е **невалиден** ако няма **Name** и **FacultyNumber**

```
class Student
{
    private string name, facultyNumber;
    public Student(string name, string facultyNumber)
    { ... }
}
```



# Променливи – други съвети

- Не дефинирайте **неизползвани променливи**
  - Компиляторът обикновено извежда предупреждение
- Не ползвайте променливи със **скрита цел**
  - Лош пример:



```
int mode = 1;  
...  
if (mode == 1) ...; // Read  
else if (mode == 2) ...; // Write  
else if (mode == 3) ...; // Read and write
```



- Вместо това ползвайте **изброяване**:

```
enum ResourceAccessMode { Read, Write, ReadWrite }
```



# Връщане на резултат от метод

- Винаги присвоете резултата на метод на **променлива** преди да го върнете. Плюсове:

- Подобрява **четливостта** на кода
  - Връщаната стойност има самоописателно име
- Опростява дебъгването

- Пример:

```
int salary = days * HoursPerDay * ratePerHour;  
return salary;
```

Функцията на формулата  
е очевидна

- Лош пример:

```
return days * HoursPerDay * ratePerHour;
```

Тук може да сложим точка на  
прекъсване и да проверим дали  
резултатът е верен



# Обхват и видимост на променливи

- Винаги опитвайте максимално да намалите обхвата и видимостта на променливите
  - Това намалява потенциалната зависимост
  - Избягвайте `public` полета (изключения: `readonly` / `const`)
  - Достъпвайте всички полета чрез `свойства` / `методи`

# Надхвърлен обхват – Пример

```
public class Globals
{
    public static int state = 0;
}
public class ConsolePrinter
{
    public static void PrintSomething()
    {
        if (Globals.state == 0)
        {
            Console.WriteLine("Hello.");
        }
        else
        {
            Console.WriteLine("Good bye.");
        }
    }
}
```



# Най-добри практики при променливите

- Инициализирайте променливи използвани в цикъл непосредствено преди него
- Не задавайте на променлива стойност докато не трябва да я използвате
  - Не следвайте стария C / Pascal стил на деклариране на променливи в началото на всеки метод
- Започнете с най-ограничената видимост
  - Разширете видимостта само при необходимост
- Групирайте свързани изрази заедно

# Групиране на свързани изрази – пример

- Шест променливи само в този кратък фрагмент

```
void SummarizeData(...)
{
    ...
    GetOldData(oldData, numOldData);
    GetNewData(newData, numNewData);
    totalOldData = Sum(oldData, numOldData);
    totalNewData = Sum(newData, numNewData);
    PrintOldDataSummary(oldData, totalOldData);
    PrintNewDataSummary(newData, totalNewData);
    SaveOldDataSummary(totalOldData, numOldData);
    SaveNewDataSummary(totalNewData, numNewData);
    ...
}
```



Трябва да следите **oldData**, **newData**, **numOldData**, **numNewData**, **totalOldData** и **totalNewData**




# По-добро групиране – пример

```
void SummarizeDaily( ... )  
{  
    GetOldData(oldData, numOldData);  
    totalOldData = Sum(oldData, numOldData);  
    PrintOldDataSummary(oldData, totalOldData);  
    SaveOldDataSummary(totalOldData, numOldData);  
    ...  
    GetNewData(newData, numNewData);  
    totalNewData = Sum(newData, numNewData);  
    PrintNewDataSummary(newData, totalNewData);  
    SaveNewDataSummary(totalNewData, numNewData);  
    ...  
}
```



Всеки от двата блока е по-кратък и съдържа по-малко променливи

# Единствена цел

- Променливите трябва да имат една-единствена цел
  - **Никога** не ползвайте една променлива за **много цели**!
  - Пестенето на памет не е извинение
- Можете ли да измислите добро име на променлива, използвана с няколко цели?
  - Пример: променлива, използвана да изброи ученици или да съхранява средния им успех
  - Предложението: **studentsCountOrAvgGrade** 

# Избягвайте сложни изрази

- Никога не ползвайте сложни изрази в кода!
- Лош пример:

```
for (int i = 0; i < xCoords.Length; i++)  
    for (int j = 0; j < yCoords.Length; j++)  
        matrix[i][j] =  
            matrix[xCoords[FindMax(i) + 1][yCoords[FindMin(j) - 1]] *  
            matrix[yCoords[FindMax(j) + 1][xCoords[FindMin(i) - 1]];
```

Какво ще правим като стигнем реда  
**IndexOutOfRangeException?**



Има 10 потенциални източника на **IndexOutOfRangeException**.

- Сложните изрази са лоши, защото:
  - Затрудняват **четенето** и **разбирането** на кода, трудни са за **дебъгване**, **промяна** и **поддръжка**

# Опростиране на сложни изрази

```
for (int i = 0; i < xCoords.Length; i++)  
{  
    for (int j = 0; j < yCoords.Length; j++)  
    {  
        int minXStartIndex = FindMin(i) - 1;  
        int maxXStartIndex = FindMax(i) + 1;  
        int minYStartIndex = FindMin(j) - 1;  
        int maxYStartIndex = FindMax(j) + 1;  
        int minXcoord = xCoords[minXStartIndex];  
        int maxXcoord = xCoords[maxXStartIndex];  
        int minYcoord = yCoords[minYStartIndex];  
        int maxYcoord = yCoords[maxYStartIndex];  
        int newValue =  
            matrix[maxXcoord][minYcoord] *  
            matrix[maxYcoord][minXcoord];  
        matrix[i][j] = newValue;  
    }  
}
```





# Избягвайте „мистериозни“ числа (Magic Numbers)

- Какво е „мистериозно“ число или стойност?
  - „Мистериозни“ числа / стойности са всички литерали, различни от **0**, **1**, **-1**, **null** и **""** (празен низ)
- Избягвайте използването им
  - Те са трудни за поддръжка
    - При промяна може да трябва да коригирате навсякъде където се появява това „мистериозно“ число / константа
  - Значението им не е очевидно
    - Пример: какво значи числото **1024**?



# Злите „мистериозни“ числа

```
public class GeometryUtils
{
    public static double CalcCircleArea(double radius)
    {
        double area = 3.14159206 * radius * radius;
        return area;
    }
    public static double CalcCirclePerimeter(double radius)
    {
        double perimeter = 6.28318412 * radius;
        return perimeter;
    }
    public static double CalcEllipseArea(double axis1, double axis2)
    {
        double area = 3.14159206 * axis1 * axis2;
        return area;
    }
}
```



# Превръщане на „мистериозните“ числа в константи

```
public class GeometryUtils
{
    public const double PI = 3.14159206;

    public static double CalcCircleArea(double radius)
    {
        double area = PI * radius * radius;
        return area;
    }

    public static double CalcCirclePerimeter(double radius)
    {
        double perimeter = 2 * PI * radius;
        return perimeter;
    }

    public static double CalcEllipseArea(double axis1, double axis2)
    {
        double area = PI * axis1 * axis2;
        return area;
    }
}
```



# Константи в C#

- В C# има два типа константи

- Compile-time константи:

```
public const double PI = 3.14159206;
```



- Заменят се със стойността си по време на компилация
- Зад тях не стои никакво поле

- Run-time константи:

```
public static readonly string ConfigFile = "app.xml";
```



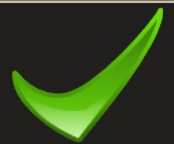
- Специални полета, инициализирани в static конструктора
- Компилират се в асемблите като всеки друг член на клас



# Кога да ползваме константи?

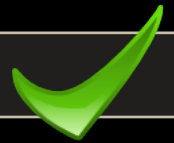
- Константи трябва да се използват в следните случаи:
  - Когато трябва да използваме имена или други стойности и техните логически смисъл и стойност не са очевидни
  - Имена на файлове

```
public static readonly string SettingsFileName =  
    "ApplicationSettings.xml";
```



- Математически константи

```
public const double E = 2.7182818284;
```



- Граници и диапазони

```
public const int ReadBufferSize = 5 * 1024 * 1024;
```



# Кога да избягваме константи?

- Понякога е по-добре да си останем с твърдо закованата стойност вместо да ползваме константа
  - Съобщения за грешки и описания на изключения
  - SQL команди за операции с бази данни
  - Имена на GUI елементи (етикети, бутони, менюта, диалози)
- За интернационализация ползвайте ресурси, не константи
  - Ресурсите са специални файлове, вградени в асемблито
  - Достъпни са по време на изпълнение на програмата

# Обобщение

1. Инициализиране на променливи
2. Съвети за използване на променливи
  - Използвайте ги да покажете целта на кода
    - Напр. когато връщате стойност от метод
  - Дръжте обхвата и живота им малки
3. Опростявайте изразите, за да избегнете трудно дебъгване
4. Ползвайте константи, за да няма „мистериозни“ стойности



# Използване на променливи, изрази и константи



Въпроси?





# Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма **"Обучение за ИТ кариера"** на МОН за подготовка по професия "Приложен програмист"



Министерство  
на образованието  
и науката



Национална  
програма  
„Обучение за  
ИТ кариера“

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**



SoftUni  
Foundation

