Отчет по лабораторной работе № 5

“Исследование изменяемых и неизменяемых типов данных языка С#”

по дисциплине

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Выполнил студент группы ИВТ/б-22о

Горбенко К.Н.

Проверил:

Тимофеев И.С.

1. **Цель работы:** исследование изменяемых и неизменяемых типов данных языка С#.
2. **Задание на работу:** учитывая, что в языке С# нет поддержки механизма дружественности, по заданию преподавателя необходимо изучить изменяемые (mutable) и неизменяемые (immutable) типы данных языка C#. Кроме того, необходимо привести примеры неудачного использования изменяемых и неизменяемых типов данных.
3. **Пример № 1.** Простейший пример неправильного использования неизменяемого типа данных - большое количество операций над типом string. При n операций конкатенации будет создано n новых объектов в куче, каждый из которых, кроме последнего, останется без указателя и впоследствии будет удален сборщиком мусора. Следующий пример несколько упрощен, т.к. при фиксированном количестве операций склеивания компилятор может оптимизировать все операции склеивания в одну:

|  |
| --- |
| var line = " "; for (int i = 1; i <= 100; i++)  line += $"{i}"; |

Данная проблема решается использованием вместо структуры string класса StringBuilder, который оптимизирован под большое количество операций:

|  |
| --- |
| var builder = new StringBuilder("");  for (int i = 1; i <= 100; i++)  builder.Append($"{i}");  var line = builder.ToString(); |

1. **Пример № 2.** Допустим, есть следующие структура и класс:

|  |
| --- |
| public struct Point  {  public int X { get; private set; }  public int Y { get; set; }   public Point(int x, int y)  {  X = x;  Y = y;  }   public void IncrementX()  {  X++;  }  }   public class Container  {  public Point Point { get; private set; }   public Container(Point point)  {  Point = point;  }  } |

Обращение к структуре в составе класса происходит через свойство класса Container:

|  |
| --- |
| Container a = new Container(new Point(5, 5));  a.Point.Y++; |

Этот код не скомпилируется, так как компилятор поймет, что произойдет изменение копии, то есть временного объекта. Но скомпилируется следующее обращение:

|  |
| --- |
| Console.WriteLine("Исходное значение Point.X: {0}", a.Point.X);  a.Point.IncrementX(); Console.WriteLine("Point.X после вызова IncrementX(): {0}", a.Point.X); |

В этом случае свойство Point вернет лишь копию, которая не будет модифицирована. То же самое произойдет, если вместо свойства класса Container использовать readonly поле, содержащее структуру (публичное поле - упрощение для простоты примера):

public class Container  
 {  
 public readonly Point point = new Point(5, 5);  
 }

В этот раз это произойдет потому, что, по соглашению, при обращении к полю только для чтения вне конструктора, генерируется временная переменная.

Использование неизменяемого значимого типа не предохраняет от создания и модификации копии, но заставляет сознательно использовать копию, если необходима структура с другим значением. Это значит, что вы не будете пойманы за модификацией копии, думая, что модифицируете оригинал.

1. **Пример № 3.** Не рекомендуется делать неизменяемыми структуры и классы, которые имеют большие объемы данных, так как при изменении их состояния в памяти произойдет снижение производительности, например:

|  |
| --- |
| public class Vehicle  {  private readonly string model;  private readonly string engineType;  private readonly double weigth;  private readonly double power;  private readonly double maxSpeed;  private readonly Unit[] units;  private readonly VolumeModel volumeModel;   public Vehicle(string model, string engineType, double weigth, double power, double maxSpeed, Unit[] units, VolumeModel volumeModel)  {  if (string.IsNullOrWhiteSpace(model)) throw new ArgumentNullException(nameof(model));  if (string.IsNullOrWhiteSpace(engineType)) throw new ArgumentNullException(nameof(engineType));  if (units == null) throw new ArgumentNullException(nameof(units));  if (volumeModel == null) throw new ArgumentNullException(nameof(volumeModel));   this.model = model;  this.engineType = engineType;  this.weigth = weigth;  this.power = power;  this.maxSpeed = maxSpeed;  this.units = units;  this.volumeModel = volumeModel;  }  }  public class Unit  {  private string name;  private string path;  }   public class VolumeModel  {  double[,] model = new double[500, 500];  } |

В данном случае, при необходимости изменить состояние объекта, потребуется создать новый объект и проинициализировать в конструкторе все его поля.

1. **Вывод.** В ходе лабораторной работы были изучены неизменяемые типы языка C#. Обнаружилось, что для изменения состояния неизменяемого типа необходимо создать новый объект. Также, была изучена проблема передачи копии изменяемой структуры при вызове через свойство или readonly поле.