|  |  |
| --- | --- |
|  | Рефакторинг кода  static bool ShouldFire(bool enemyInFront, string enemyName, int robotHealth)  {  bool shouldFire = true;  if (enemyInFront == true)  {  if (enemyName == "boss")  {  if (robotHealth < 50) shouldFire = false;  if (robotHealth > 100) shouldFire = true;  }  }  else  {  return false;  }  return shouldFire;  } |

Посмотрев на код выше, сразу бросается количество ветвлений условных операторов. И относительно не большая функциональность метода, занимает слишком много строк кода.

|  |
| --- |
| static bool ShouldFire2(bool enemyInFront, string enemyName, int robotHealth)  {  return enemyInFront? ((enemyName=="boss")&& robotHealth<50?false:true):false;  } |

Метод настолько уменьшился что поместился в одну строку, а функциональность его осталось не изменой.

Есть целый ряд статей и сообщений, которые говорят , что StringBuilder является более эффективным , поскольку он содержит изменяемый буфер строки. .NET Строки неизменны, что является причиной , почему новый String объект создается каждый раз , когда мы изменяем его (вставка, Append, удалить и т.д.).

Как вы можете видеть, я определила String переменную и StringBuilder переменную. Затем добавила строку в обоих этих переменных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;    namespace StringTest  {      class Program      {          static void Main(string[] args)          {              string s = "Pranay";              s += " rana";              s += " rana1";              s += " rana122";                StringBuilder sb = new StringBuilder();              sb.Append("pranay");              sb.Append(" rana");          }      }  } |

Если разобрать этот код под рефлетором, станет понятно по какой причине лучше использовать StringBuilder.

Если вы зашли в рефлектор можете видеть, Concat функция принимает два аргумента и возвращает String. Следующие шаги выполняются , когда мы выполняем Append с типом строки:

1. Проверяет, является ли строка пустой или нет
2. Создает строку destи выделяет память для строки
3. Заполняет destстроку str0иstr1
4. Возвращает destстроку, которая представляет собой новую переменную строку

Таким образом, это доказывает, что всякий раз, когда я делаю операцию конкатенации строк, он создает новую строку из-за неизменного поведения строк.