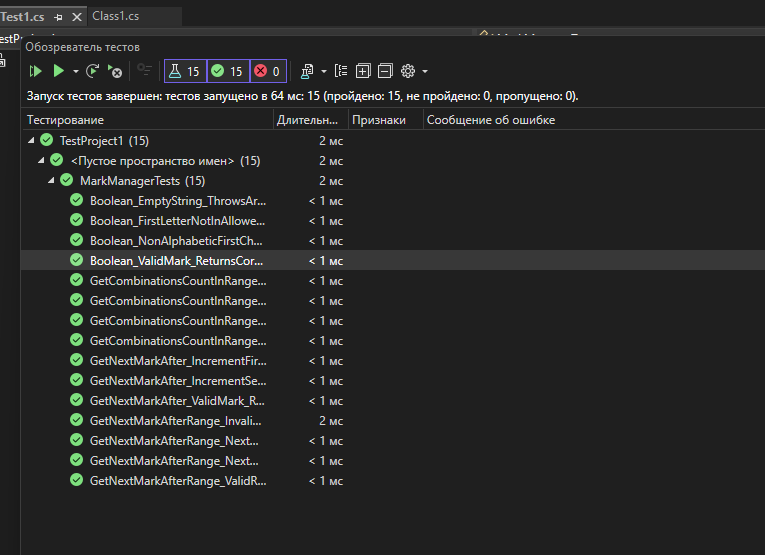
|  |
| --- |
| Class1 class1 = new Class1();  [TestMethod]  public void Boolean\_FirstLetterNotInAllowedSet\_ReturnsFalse()  {  Assert.IsTrue(class1.Boolean("C123BC456"));  }  [TestMethod]  public void Boolean\_EmptyString\_ThrowsArgumentException()  {  Assert.Throws<ArgumentException>(() => class1.Boolean(""));  }  [TestMethod]  public void Boolean\_NonAlphabeticFirstCharacter\_ThrowsArgumentException()  {  Assert.Throws<ArgumentException>(() => class1.Boolean("1A23BC456"));  }  [TestMethod]  public void GetNextMarkAfter\_IncrementSecondLetter\_ReturnsNextMark()  {  Assert.AreEqual("A123BC457", class1.GetNextMarkAfter("A123BC456")); // Проверка на инкремент второй буквы  }  [TestMethod]  public void GetNextMarkAfter\_IncrementFirstLetter\_ReturnsNextMark()  {  Assert.AreEqual("A123BD000", class1.GetNextMarkAfter("A123BC999")); // Проверка на инкремент первой буквы  }  [TestMethod]  public void GetNextMarkAfterRange\_NextMarkOutsideRange\_ThrowsArgumentException()  {  Assert.Throws<ArgumentException>(() => class1.GetNextMarkAfterRange("A123BC456", "A123BC000", "A123BC455")); // Проверка выхода за пределы диапазона  }  [TestMethod]  public void GetNextMarkAfterRange\_NextMarkWithinRange\_ReturnsNextMark()  {  Assert.AreEqual("A123BC457", class1.GetNextMarkAfterRange("A123BC456", "A123BC000", "A123BC999")); // Проверка на корректный следующий знак  }  //  [TestMethod]  public void GetCombinationsCountInRange\_SameMarks\_ReturnsOne()  {  Assert.AreEqual(1, class1.GetCombinationsCountInRange("A123BC456", "A123BC456")); // Проверка на одинаковые знаки  }  [TestMethod]  public void GetCombinationsCountInRange\_LargeRange\_ReturnsCorrectCount()  {  Assert.AreEqual(1000, class1.GetCombinationsCountInRange("A123BC000", "A123BC999")); // Проверка большого диапазона  }  [TestMethod]  public void GetCombinationsCountInRange\_InvalidMarks\_ThrowsArgumentException()  {  Assert.Throws<ArgumentException>(() => class1.GetCombinationsCountInRange("A123BC999", "A123BC000")); // неправильный порядок  }  [TestMethod]  public void GetNextMarkAfterRange\_InvalidRange\_ThrowsArgumentException()  {  Assert.Throws<ArgumentException>(() => class1.GetNextMarkAfterRange("A123BC456", "A123BC500", "A123BC400")); // неправильный диапазон  }  [TestMethod]  public void Boolean\_ValidMark\_ReturnsCorrectBoolean()  {  Assert.IsTrue(class1.Boolean("A123BC456")); // A - допустимый  Assert.IsTrue(class1.Boolean("B123BC456")); // B - допустимый  Assert.IsTrue(class1.Boolean("C123BC456")); // C - недопустимый  }  // Проверка на получение следующего знака после определенного  [TestMethod]  public void GetNextMarkAfter\_ValidMark\_ReturnsNextMark()  {  Assert.AreEqual("A123BC457", class1.GetNextMarkAfter("A123BC456")); // Пример получения следующего знака  }  // Проверка на корректное количество комбинаций в диапазоне  [TestMethod]  public void GetCombinationsCountInRange\_ValidRange\_ReturnsCorrectCount()  {  Assert.AreEqual(1000, class1.GetCombinationsCountInRange("A123BC000", "A123BC999")); // Пример количества комбинаций  }  // Проверка на корректное получение следующего знака в диапазоне  [TestMethod]  public void GetNextMarkAfterRange\_ValidRange\_ReturnsNextMark()  {  Assert.AreEqual("A123BC457", class1.GetNextMarkAfterRange("A123BC456", "A123BC456", "A123BC500")); // Пример  } |



1. Boolean\_FirstLetterNotInAllowedSet\_ReturnsFalse
   * Проверяет, что метод возвращает true, если первая буква знака находится в допустимом наборе (например, "C123BC456"). В данном случае "C" является допустимой буквой.
2. Boolean\_EmptyString\_ThrowsArgumentException
   * Проверяет, что метод выбрасывает ArgumentException, если передана пустая строка. Это важно для гарантии, что метод не будет работать с некорректными входными данными.
3. Boolean\_NonAlphabeticFirstCharacter\_ThrowsArgumentException
   * Проверяет, что метод выбрасывает ArgumentException, если первая буква знака не является буквой (например, "1A23BC456"). Это необходимо для обеспечения корректности формата знака.
4. GetNextMarkAfter\_IncrementSecondLetter\_ReturnsNextMark
   * Проверяет, что метод корректно инкрементирует второй символ знака (например, "A123BC456" становится "A123BC457"). Это подтверждает правильность логики для получения следующего знака.
5. GetNextMarkAfter\_IncrementFirstLetter\_ReturnsNextMark
   * Проверяет, что метод корректно инкрементирует первую букву знака, когда все числовые символы достигли своего максимума (например, "A123BC999" становится "A123BD000"). Этот тест важен для проверки логики перехода между буквами.
6. GetNextMarkAfterRange\_NextMarkOutsideRange\_ThrowsArgumentException
   * Проверяет, что метод выбрасывает ArgumentException, если следующий знак находится за пределами указанного диапазона (например, "A123BC456" с диапазоном "A123BC000" и "A123BC455"). Это гарантирует, что метод не вернет некорректные значения.
7. GetNextMarkAfterRange\_NextMarkWithinRange\_ReturnsNextMark
   * Проверяет, что метод возвращает следующий знак, находящийся в указанном диапазоне (например, "A123BC456" с диапазоном "A123BC000" и "A123BC999" возвращает "A123BC457"). Это подтверждает корректную работу метода.
8. GetCombinationsCountInRange\_SameMarks\_ReturnsOne
   * Проверяет, что метод возвращает 1, если начальный и конечный знаки совпадают (например, "A123BC456" и "A123BC456"). Это важно для проверки логики подсчета комбинаций.
9. GetCombinationsCountInRange\_LargeRange\_ReturnsCorrectCount
   * Проверяет, что метод возвращает правильное количество комбинаций в большом диапазоне (например, от "A123BC000" до "A123BC999" возвращает 1000). Это важно для проверки работоспособности метода при больших диапазонах.
10. GetCombinationsCountInRange\_InvalidMarks\_ThrowsArgumentException
    * Проверяет, что метод выбрасывает ArgumentException, если начальный знак больше конечного (например, "A123BC999" и "A123BC000"). Это необходимо для предотвращения некорректных данных.
11. GetNextMarkAfterRange\_InvalidRange\_ThrowsArgumentException
    * Проверяет, что метод выбрасывает ArgumentException, если передан неверный диапазон (например, "A123BC456" с диапазоном "A123BC500" и "A123BC400"). Это гарантирует, что метод работает только с корректными диапазонами.
12. Boolean\_ValidMark\_ReturnsCorrectBoolean
    * Проверяет, что метод возвращает true для допустимых знаков, таких как "A123BC456" и "B123BC456", и false для недопустимого знака "C123BC456". Это важно для проверки корректности работы метода.
13. GetNextMarkAfter\_ValidMark\_ReturnsNextMark
    * Проверяет, что метод возвращает корректный следующий знак для заданного (например, "A123BC456" возвращает "A123BC457"). Это подтверждает правильность работы метода.
14. GetCombinationsCountInRange\_ValidRange\_ReturnsCorrectCount
    * Проверяет, что метод возвращает правильное количество комбинаций в заданном диапазоне (например, от "A123BC000" до "A123BC999" возвращает 1000). Это важно для проверки работоспособности метода.
15. GetNextMarkAfterRange\_ValidRange\_ReturnsNextMark

Проверяет, что метод возвращает корректный следующий знак в пределах заданного диапазона (например, "A123BC456" с диапазоном от "A123BC456" до "A123BC500" возвращает "A123BC457"). Это важно для подтверждения того, что метод правильно обрабатывает диапазоны и возвращает ожидаемые результаты.