Звіт з “Інженерії програмного забезпечення”

з лабораторної роботи “Рефакторинг”

студентки 3-го курсу КІ-СА
Уваренко Олени

Мені випала нагода працювати із курсовою роботою, метою якої є створити графічний інтерфейс(Віндовс форми) для роботи із деревом.
Яким саме, на жаль, не відомо.

1. Code smells

Щодо запахів коду, то їх, насправді, не було виявлено. Єдиною, але дуже великою проблемою є невідповідність коду відносно логіки програми.
Наприклад, використання методу cumm_summ є абсолютно недоречним, адже його логіка неправильна, і, напревеликий жаль, застосована і у конструкторі, і у всіх методах класу дерева.

```
public int cumm_summ(int idx, int[] arr_inp)
{

    int summ = 0;
    int end = idx - (idx & -idx);

    while (idx > end)
    {
        summ += arr_inp[idx];
        idx -= 1;

    }

    return summ;
}
```

Так само, як недоречним є використання властивості-індексатору-колекції:

```
public List<int> this[int i]
{
    get
    {
        return arr_list[i];
    }
}
```

У конструкторі:

```
public Fenwick_Tree(int[] arr_inp)
{
    List<int>[] arr_list = new List<int>[arr_inp.GetLength(0)];

    for (int i = 0; i < arr_inp.GetLength(0); i++)
    {

        List<int> tempList = new List<int>();
        tempList.Add(cumm_summ(i, arr_inp));
        arr_list[i] = tempList;

    }
    this.arr_list = arr_list;

}
```

2. Модульні тести

1. Метод actual:
   Класи хороших та поганих даних, відповідно

```
[TestMethod]
public void test_actual_good_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(3, ft.actual(3));
}

[TestMethod]
public void test_actual_bad_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(false, ft.actual(3)==4);
}
```

2. Метод read:
   Класи хороших та поганих даних, відповідно

```
[TestMethod]
public void test_read_good_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(6, ft.read(3));
}

[TestMethod]
public void test_read_bad_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(false, ft.read(3)==8);
}
```

3. Метод range_read:
   Класи хороших та поганих даних, відповідно

```
[TestMethod]
public void test_range_read_good_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(9, ft.range_read(ft, 2, 4));
}

[TestMethod]
public void test_range_read_bad_data()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(false, ft.range_read(ft, 2, 4)==20);
}
```

4. Властивість Count(у ній є трохи логіки)

```
[TestMethod]
public void test_count_prop_good()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(arr_input.GetLength(0), ft.Count);
}

[TestMethod]
public void test_count_prop_bad()
{
    int[] arr_input = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
    Tree ft = new Tree(arr_input);
    Assert.AreEqual(false, ft.Count==ft.Count+1);
}
```

3. Рефакторинг

Рефакторинг був винонаний капітально. Як саме? Мені довелось переписати всю логіку класу із самого початку:
Метод read:
До рефакторингу:

```
// метод розрахунку кумулятивної суму від початку масиву до певного заданого значення
public int read(int idx)
{
    int sum = 0;
    while (idx > 0)
    {
        sum += arr_list[idx][0];
        idx -= (idx & -idx);
    }
    return sum;
}
```

Після:

```
// метод розрахунку кумулятивної суму від початку масиву до певного заданого значення
public int read(int idx)
{
    int sum = 0;

   for (int i = 0; i <= idx; i++)
    {
        sum += arr_input[i];
    }

    return sum;
}
```

Ну, і так само із усіма іншими методами:

До:

```
public int actual(int idx)
{
    int sum = arr_list[idx][0];
    if (idx > 0)
    {
        int z = idx - (idx & -idx);
        idx--;
        while (idx != z)
        {
            sum -= arr_list[idx][0];

            idx -= (idx & -idx);
        }
    }
    return sum;
}
```

Після:

```
public int actual(int idx)//витягує елемент по індексу
{

    int value = arr_input[idx];
    return value;
}
```

До:

```
// зчитування кумулятивної суми на певному відрізку
public int range_read(Tree t, int idx1, int idx2)
{
    return t.read(idx2) - t.read(idx1-1);
}
```

Після:

```
// зчитування кумулятивної суми на певному відрізку
  public int range_read(Fenwick_Tree t, int a, int b)
  {
      return t.read(b) - t.read(a - 1);
  }
```

Деякі із них, як-от метод update, наприклад, довелось видалити за їх несумісністю із програмою(мертвий код).

Висновок:

У даній лабораторній роботі я навчилась використовувати методи рефакторингу, писати модульні тести, відточила навички роботи із чужим кодом.