VI/2. Egy feladat algoritmikus és OOP megoldása

A programtervező informatikus „elméleti” tervező szemléletmódja és a mérnökinformatikus megvalósításra fókuszáló programozási „gyakorlata” más paradigmák mentén értelmezi és oldja meg a problémákat. Az Eratosztenészi szita implementációin jól érzékelhető a programtervező algoritmikus és a mérnök objektumra, adatra fókuszáló gondolkodásmódjából adódó különbség:

Példa: Eratosztenészi szita.

Adjuk meg egy adott *N* értékig a prímszámokat! [43]

Strukturált, procedurális megoldás

Vegyünk fel egy N + 1 elemű logikai tömböt, melynek 0. és 1. eleme false, a többi kezdetben true.

- Egy iterátorral menjünk végig a tömb elemein N gyökéig és minden esetben, amikor a mutatott elem true,
  - az adott helyindex többszöröseinek minden létező helyén levő értékét állítsuk false-ra.
- A prímszámok listáját a tömb true értéket tartalmazó indexei adják.

Objektum orientált megoldás

Készítsünk egy két végén strázsás, kétirányban láncolt listát, amelynek

- kezdete egy adagoló (a kapott számot tovább küldi);
- a vége a prímszámok kiírója, azaz gyűjtője, a vödör (a kapott számot kiírja és beszúr maga elé egy adott értékű szűrőt;
- a közbenső elemek az eljárás során keletkező szűrők, mely a kapott számot csak akkor küldi tovább, ha a saját értékével a szám nem osztható.

Kód (C# nyelven, *N* = 100 értékkel)

[43] OOP absztrakció forrása: C11 és C++11 programozás (VIEEAV01) tantárgy, 2. labor 1. feladat (2017/2018 II. félév, évente megújuló, belső anyag) https://cpp11.eet.bme.hu/lab02/index.html [2018.04.14]

```
using System;
namespace Eratosz_Proc
{
class Program
{
  static void Main()
  {
    bool[] prim = new bool[101];
    prim[0] = false;
    prim[1] = false;
    for (int i = 2; i <= 100; i++)
      prim[i] = true;
    int ix = 2;
    while (ix * ix <= 100)
    {
      if (prim[ix])
      {
        int m = 2;
        while (m * ix <= 100)
        {
          prim[m * ix] = false;
          m++;
        }
      }
      ix++;
    }
    for (int i = 0; i <= 100; ++i)
      if (prim[i])
        Console.Write(i + ", ");
  }
}
}
```

```
using System;
namespace Eratosz_OOP
{
  abstract class Elem {
    public Elem Prev { get; set; }
    public Elem Next { get; set; }
    public abstract void Todo(int szam);
  }
  class Adagolo : Elem {
    public override void Todo(int szam) {
      Next.Todo(szam);
    }
  }
  class Szuro : Elem {
    private readonly int ertek;
    public Szuro(int ertek) {
      this.ertek = ertek;
    }
    public override void Todo(int szam) {
      if(szam % ertek != 0) Next.Todo(szam);
  }
  class Vodor : Elem {
    public override void Todo(int szam) {
      Console.Write("{0}, ", szam);
      Szuro uj = new Szuro(szam);
      uj.Prev = this.Prev; uj.Next = this;
      this.Prev = uj; uj.Prev.Next = uj;
    }
  }
  class Program{
    static void Main()   {
      Adagolo a = new Adagolo();
      Vodor v = new Vodor();
      a.Next = v; v.Prev = a;
      for (int i = 2; i <= 100; i++) a.Todo(i);
    }
  }
}
```

Míg a procedurális megoldásban egy sorozatszámítás és egy kigyűjtés kombinációját láthatjuk, az OOP megoldásban a szűrő objektumok továbbadják egymásnak a számukra nem megfelelő számot. A vödör objektumig jutó szám a kiírás mellett egy újabb szűrőt eredményez.

4.4
6.2.4