1. feladat:

a. UserControl fogalma és jelentősége a tervezés/fejlesztés során. (3p)

- A vezérlőelem maga is egy űrlap, tartalmazhat vezérlőelemeket. Tervezési időben vizuálisan elkészíthetjük összetett vezérlelemeinket, pont úgy, ahogy egy formot is elkészítenénk. Űrlapokra, illetve más UserControlokra lehet elhelyezni.
- Újrafelhasználás, valamint összetett felhasználói felület modularizálásának eszköze

b. Üzenetkezelő ciklus megmagyarázása. (2p)

- Minden threadhez, ami ablakot kezel, tartozik egy üzenetsor
- A System Queue-ből kerülnek az üzenetek az app üzenetsorába, ahonnan az Üzenetkezelő Ciklus szedi ki, és adja az ablakkezelő függvénynek

c. C# kódot kellett írni: a 20,20 koordinátára rajzoljon ki egy 60px oldalhosszúságú telizöld négyzetet, és ennek másodpercenként csökkenjen az oldalhosszúsága 5px-el addig, amíg az x billentyű le nem lett nyomva. (12p)

```
public partial class Form1 : Form
{
   bool stopped = false;
   int i = 60;

   public Form1()
   {
      Timer timer = new Timer();
      timer.Interval = 1000;
      timer.Tick += new EventHandler(timer_Tick);
      timer.Start();
      this.KeyDown += new
KeyEventHandler(Form1_KeyDown);
      InitializeComponent();
   }

   void Form1_KeyDown(object sender, KeyEventArgs e)
   {
      if (e.KeyCode == Keys.X && !stopped)
      { stopped = true; }
   }

   void timer_Tick(object sender, EventArgs e)
   {
      Invalidate();
      if (!stopped) i -= 5;
   }

   protected override void OnPaint(PaintEventArgs e)
   {
      base.OnPaint(e);
      Brush greenpen = new SolidBrush(Color.Green);
      e.Graphics.FillRectangle(greenpen, new Rectangle(20, 20,
i, i));
         }
}
```

2. feladat:

a. Mikor érdemes többszálú alkalmazást használni? Legalább 3 alkalom. (3p)

- Átlagosan jobb CPU kihasználás elérése
- Hosszú blokkoló művelet GUI alkalmazásokban
- Időzítésérzékeny feladatok
- Kiszolgáló (szerver) alkalmazások (pl. webszerver)

b. ReaderWriterLock (vagy ReaderWriterLockSlim) és Mutex összehasonlítása. (2p)

- Mutex: Mint a lock, de folyamatok között is. Pl. annak megoldására, hogy egy alkalmazásból csak egy példány indulhasson.
- ReaderWriterLock: Sok olvasóra optimalizált megoldás. Egyszerre több olvasó is hozzáférhet az erőforráshoz, de íróból csak egy (illetve az író kizárja az olvasókat is). Pl. ritkán módosított cache megvalósítása.

c. C# kódot írni: konkurrens stack megvalósítása, egészeket tárol, void Push(int i) művelet betesz egy számot a tetejére, int Pop() pedig visszaadja a tetején lévőt (de nem törli). (9p)

```
class ThreadSafeClass
    {
        private static object syncObject = new
object();
        private Stack<int> list;

        public ThreadSafeClass()
        {
            list = new Stack<int>();
        }

       public void Push(int o)
        {
            lock (syncObject)
            {
                list.Push(o);
            }
        }
        public Object Pop()
        {
            int o = 0;
            lock (syncObject)
            {
                if (list.Count > 0)
                {
                    o = list.Pop();
                }
            }
            return o;
        }
    }
```

3. feladat:

C# kódot kellett írni: Square osztály, private int side mező, public SetSide(int newValue) metódus beállítja a mező értékét, majd elsüti a SideChanging eseményt, és ebben paraméterül visszaadja a sides mező régi és új értékét is. Ezt kellett megvalósítani, meg egy másik osztályt, ami feliratkozik az eseményére. (14p)

```
class Program
    {
        static void Main(string[] args)
        {
            Square square = new Square();

            square.SideChanging +=
square_SideChanging;

            square.SetSide(16);

            Console.ReadKey();
        }

        static void square_SideChanging(int
newValue, int oldValue)
        {
            Console.WriteLine(newValue + " " +
oldValue);
        }
    }
```

```
    class Square
    {
        private int side;
        //  Az esemenyek tipusanak definialasa
        public delegate void
SideChangingEventHandler(int newValue, int
oldValue);

        //  Az esemeny definicioja
        public event SideChangingEventHandler
SideChanging;

        public void SetSide(int newValue)
        {
            if(SideChanging!=null)
                SideChanging(newValue, side);
            side = newValue;
        }
    }
```

Maradék:

55p összesen

4. feladat:

ADO.NET kapcsolatalapú modellel a Movie(MovieID, Title, Rating) táblából kitörölni azokat a sorokat, ahol a Rating < 7 vagy nincs megadva. (13p)

```
SqlConnection conn = null;

        try
        {
            conn = new SqlConnection("Data
Source=.\\SQLEXPRESS;Initial Catalog=SzoftechDB;Integrated
Security=True");
            conn.Open();

            SqlCommand command = new
SqlCommand("DELETE FROM Movie(MovieID, Title, Rating)
WHERE Rating < '7' OR Rating IS NULL");
            command.Connection = conn;
            command.ExecuteNonQuery();
        }

        catch (Exception ex) { Console.WriteLine(ex.Message);
}

        finally
        {
            if ((conn != null) && (conn.State ==
System.Data.ConnectionState.Open))
                conn.Close();
        }
```

5. feladat:

a. UML osztálydiagramból kellett C#, Java, vagy C++ kódot írni. (6p)

b. Leírtak egy történetet, hogy te vagy a vezető szoftverfejlesztő egy cégnél, ahol képkezelő alkalmazást készítenek C#-ban. A képtípusokat (PngImage, GifImage, ...) összefogták egy közös ős, az ImageBase alá, hogy elkerüljék a kódduplikálást. Szeretnék a JPEG támogatást is hozzáadni, erre az egyik alkalmazott talált egy küldő dll-t, ami tartalmaz egy JpegBase absztrakt osztályt, ami jó lenne a Jpeg képek ősének. Kritikus döntést kell hoznod: hogyan módosítanád a jelenlegi struktúrát úgy, hogy a dll-t fel lehessen használni? A dll nyilván nem módosítható (nem tudod leszármaztatni osztályból, nem valósíthat meg interfészt, de pl. belőle származhatnak le). A megoldást meg is kellett magyarázni. (9p)

6. feladat:

a. A C++ template-ek három nagy hátránya, ami .NET-ben nincs. (3p)

- A sablonok fordításkor fejtődnek ki, de csak a felhasználás során! Ha nem használunk egy sablont, ki se derülnek a benne levő hibák.

Minden sablonparaméter-kombinációra külön kód generálódik!

- Kódburjánzás (code bloat) veszély!
- A sablon forráskódja a felhasználás során a fordításkor rendelkezésre kell álljon! A forráskód védelme nem megoldott!

b. Reflexió fogalma, 3 szolgáltatása. (3p)

- Lekérdezhetjük, hogy egy szerelvényben milyen típusok vannak
- Lekérdezhetjük: az egyes típusok (osztályok, interfészek, stb). felépítését: pl. tagváltozók, tagfüggvények, event-ek, stb. listája. Be is állíthatjuk a tagváltozók értékét, meg is hívhatjuk az egyes metódusokat
- Lekérdezhetjük az egyes nyelvi elemekhez (osztályok, tagjaik, stb.) tartozó attribútumokat

c. Reflexiós példakód: változó típusának kiírása. (4p)

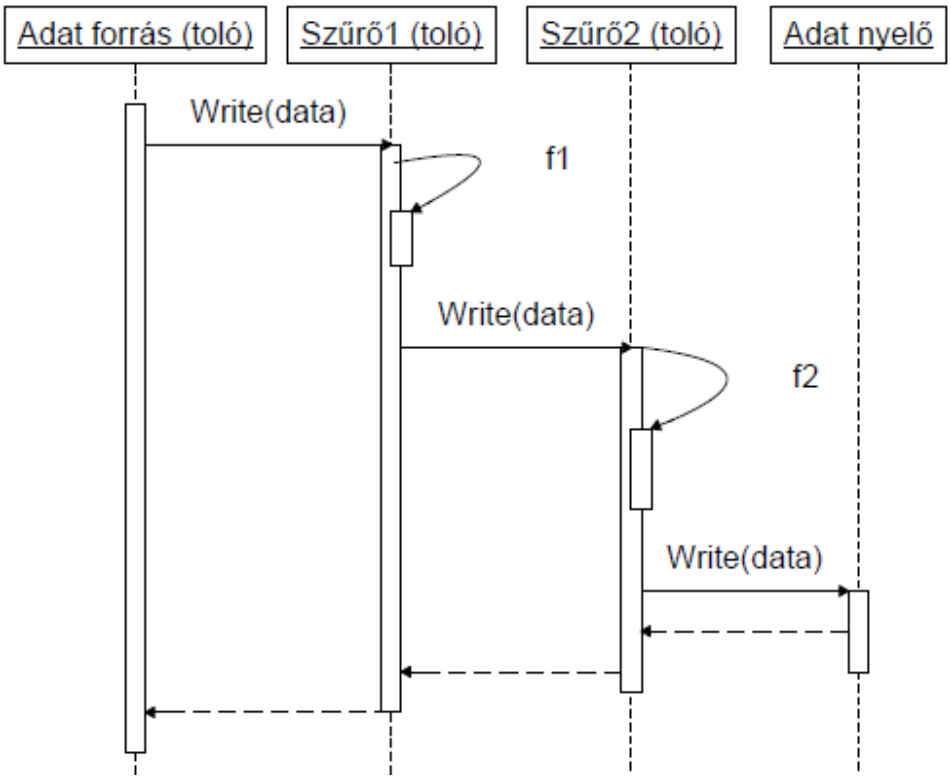
```
Complex c1 = new Complex(10, 10);
Type t1 = c1.GetType();
Console.WriteLine(t1.FullName);
```

d. Reflexiós példakód: osztály tagváltozóinak a neveinek a kiírása. (4p)

```
Type type = typeof (Complex);
Console.WriteLine("Fields:");
foreach (FieldInfo fi in type.GetFields())
        Console.WriteLine("\t" + fi.Name);
```

7. feladat:

a. Adatforrás által vezérelt csővezeték szekvencia diagrammal + magyarázattal. (8p)



b. Csővezetékbe beépíthető szűrő pszeudokód. (6p)

void Write(Data data)

{ Data processedData = ProcessData(data);

nextFilter.Write(processedData); }