1. Óra

Elérhetőség

Márton Balázs

- g68pwv@inf.elte.hu
- Teams (preferált)
- Tantárgy honlapja: https://people.inf.elte.hu/gt/oep/
  - Korábbi ZH-k
  - Algoritmus minták
  - Feladatsorok

Bevezető

Teljesítéshez szükséges feltételek

TEENDŐ	***MINIMUM***
5 kötelező beadandó	határidőre, mind az 5 elfogadható
5 szorgalmi beadandó	-
1 nagy beadandó	több lépésben határidőre, min 2-es
1 kódolós ZH	min 3-as (1,3,5 jegyeket lehet szerezni), *1x javítható*
10 EA kvíz	*min 75%*
+/- kérdések	*min 50%*
2 tervezős ZH	*mindkettő min 2-es, az egyik 1x javítható*

Beadandók elosztása

Kettőnként csoportosítva, mindegyik 2-ből egyet legalább meg kell oldani.

Beadandók beadása

Ami ***NEM***:

- *obj/* mappa (lefordított objektumok)
- *bin/* mappa (buildelt bináris fileok)

ℹ️ Ezek buildelésnél újrafordulnak, felesleges ezzel tölteni a szervereket

Ami kell:

- Solution file (*.sln*)
- Project file (*.csproj*)
- C# class fileok (*.cs*)

ℹ️ Ezeket zippelve

Nagy beadandó ütemezése

Tervezős gyakorlatvezetőnek:

- *Specifikáció* (használati esetek, objektum diagram, néhány szekvencia diagram) a **7. hétig**
- *Modell* (részletes osztály diagram, de tervezési minták nélkül) a **11. hétig**
- *Teljes tervdokumentáció* (specifikáció, osztálydiagram, állapotgépdiagram) a **13. hétig**

Kódolós gyakorlatvezetőnek:

- *Teljes megoldást* (lekódolt verzió) szorgalmi időszak végén egy határidőig (TBD)
  - Ha elfogadott akkor az *5-öst* ér
  - Minden hét késés után *-1 jegy*

Jegy kiszámítása

```
( (tervezős zh-k jegyei + beadandó jegye + kódolós zh jegye) / 4 ) + 0.1 * szorgalmik száma
```

egészre kerekített értéke

Tematika

Óra	*Kódolós gyakorlat tematika (kb)*
1	osztály, metódus, konstruktor, láthatóság, enum, getter/setter, névtér, static operator, kivétel-kezelés
2	tömb vs. List<T>, stack, heap, overloading, overriding, operátor definíció
3	indexelő property, algoritmus minták kódja, automatikus tesztkörnyezet
4	menü
5	algoritmus minták szekvenciális inputfájlon, nyomkövetés, szürke doboz tesztelés
6	felsorolható objektumok, struct/class/record
7	1-n kapcsolat, szerepnevek a kódban
8	altípusos polimorfizmus, futásidejű polimorfizmus, abstract, virtual metódusok
9	runtime típusellenőrzés, castolás
10	interface, tervminták
11	osztálydiagram megvalósítása, felpopulálás fájlból
12	generikusok, állapotgép
13	multithreading, állapotgép

Pár egyéb dolog

- Beadandókat beadni → Canvasra
- Órai munkák, jegyzetek → Teamsen **#22 csoport - Files**
- Plágium: ellenőrzés van, ne plz, not cool
- Órái menetrend
  - bevezetés
  - kódolás (elsősorban gyakorlatok feladatait)

ℹ️ Tárgy lényege: modell elkészítése, implementálás elkülönítése ⟹ 2 féle gyakorlat

- Nem a nyelv a lényeg, hanem az objektumelvűség
- A kódolásnál „alulról-felfelé”: először az osztályokat, majd a főprogramot készítjük el

---

Miről lesz szó

Osztály (Class)

ℹ️ Az objektumok *"blueprintje"*

ℹ️ Definíció az objektumok implementálására

⚠️ Speciális típus (később bővebben)

```
public class Dispenser { ... }
```

Konstruktor (Constructor)

ℹ️ Az osztály konstruktora példányosításakor fut le

ℹ️ Inicializáláshoz szükséges kód (pl: privát adattagoknak értékadás, ezek validálása)

`Dispenser.cs`

```
public class Dispenser {
    private int _tele;  //
    private int _adag;  // Adattagok (Private Fields) deklarálása
    private int _akt;   //

    public Dispenser(int k, int e) { // Konstruktor
        _tele = k;  //
        _adag = e;  // értékadás
        _akt = 0;   //
    }
}
```

`Program.cs`

```
public class Program {
    static void Main(string[] args) {
        // Példányosítás, új instance, objektum
        Dispenser MyLittleDispenser = new Dispenser(10, 1);

        // Példányosítás, új instance, objektum
        Dispenser MyNotSoLittleDispenser = new Dispenser(80, 1);
    }
}
```

What is this?

ℹ️ A class adott példányát (instance) jelenti

```
public class Dispenser {
    private int tele;   //
    private int adag;   // Adattagok deklarálása
    private int akt;    //

    public Dispenser(int tele, int adag) { // Konstruktor
        this.tele = tele;  //
        this.adag = adag;  // értékadás
        this.akt = 0;      //
    }
}
```

Láthatóság (Visibility)

```
public double trouble;            // mindenhonnan
protected internal byte of87;     // ugyanazon a projekten belül + leszármazott
osztályban (még ha más projektben van is)
protected int eget;               // ugyanabban az osztályban + leszármazott osztályban
internal class Program { ... }    // ugyanazon a projekten belül
private protected ushort shorty;  // ugyanabban az osztályban + leszármazott osztályban
(de ugyanabban a projektben)
private long bottom;              // ugyanabban az osztályban
```

Néhány dolog default láthatósága:

`public`

- Enumok

`internal`

- Osztályok
- Interfacek

`private`

- Adattagok
- Lokális változók

Getter, Setter

ℹ Osztályok adattagjainak olvasása, módosítása osztályon kívül

`Dispenser.cs`

```
public class Dispenser {
    private int _tele;  //
    private int _adag;  // Adattagok (Private Fields)
    private int _akt;   //

    // Tulajdonságok (Public Properties) (Accessors)
    // Adattagokat elérhetővé tesznek írásra / olvasásra
    public int Tele { get { return _tele; } private set; }
    public int Akt { get { return _akt; } set { _akt = value; } }
```

```
    public Dispenser(int k, int e) { // Konstruktor
        this._tele = k;
        this._adag = e;
        this._akt = 0;
    }
}
```

`Program.cs`

```
public class Program {
    static void Main(string[] args) {
        // Példányosítás, új instance, objektum
        Dispenser MyLittleDispenser = new Dispenser(10, 1);

        // OK, van publikus getterje
        int tele = MyLittleDispenser.Tele;

        // nope, ez egy privát adattag
        int privateTele = MyLittleDispenser._tele;

        // nope, nincs publikus setterje
        MyLittleDispenser.Tele = 4;

        // OK, van publikus setterje
        MyLittleDispenser.Akt = 5;
    }
}
```

Megjegyzések

```
public int Tele { get { return _tele; } private set; }
```

⟺

```
public int Tele { get { return _tele; } }
```

⟺

```
public int Tele => _tele;
```

Metódus (Method)

ℹ Objektumhoz tartozó függvény

`Dispenser.cs`

```
public class Dispenser {
    ...

    public void Dispense() { // Publikus Metódus
        Console.WriteLine("Dispensing stuff");
```

```
    }

    public bool DispenseI(int i) { // Paraméterekkel és visszatérési értékkel
        Console.WriteLine($"Dispensing {i} stuff");
        return True;
    }
}
```

`Program.cs`

```
public class Program {
    static void Main(string[] args) {
        // Példányosítás, új instance, objektum
        Dispenser MyLittleDispenser = new Dispenser();

        // Metódus hívás
        MyLittleDispenser.Dispense();

        // Metódus hívás
        bool returnedValue = MyLittleDispenser.DispenseI(42);
    }
}
```

Osztályszintű metódusok, static

- nem kell ahhoz példányosítani az osztályt, hogy használhassuk
- statikus metódus nem-statikus adattagokhoz (és bármi egyébhez, ami a példányhoz kötött) nincs hozzáférése (mert nincs referencia az objektumra)

```
public class Calculator {
    public static int Add(int x, int y) {
        return x + y;
    }
}

public class Program {
    static void Main(string[] args) {
        // Itt nincs példányosítás!
        int result = Calculator.Add(6, 9); // 15
    }
}
```

```
class Test {
    private int _x = 0;
    private static int _y = 0; // osztályszintű adattag

    public static int Foo() {
        return _x; // Ez NEM működik.
    }
    public static int Bar() {
        return _y; // Ez viszont igen!
    }
}
```

```
enum CatTypes { Orange, NonOrange }

public class Cat {
    ...
    public CatTypes Type { get; set; }

    public int Braincells => Calculator.CalculateBraincells(this);
}

public class Calculator {
    public static int CalculateBraincells(Cat cat) {
        if (cat.Type == CatTypes.Orange) {
            return 1;
        }
        ...
    }
}
```

Névtér (Namespace) (kitérő)

ℹ Scopeok kontrollálása, osztályok csoportosítása, szervezése (Scope: változók elérhetősége a kód különböző pontjain)

```
namespace FirstSpace {
    public class First {
        public static void Greet() {
            Console.WriteLine("Greetings from the first space!");
        }
    }
}

namespace SecondSpace {
    public class Second {
        public static void Greet() {
            Console.WriteLine("Greetings from the second space!");
        }
    }
}

// Beágyazott namespace-ek
namespace Outer {
    public class OuterClass {
        public static void Greet() {
            Console.WriteLine("Greetings from outer space! 🚀");
        }
    }

    namespace Inner {
        public class InnerClass {
            public static void Greet() {
                Console.WriteLine("Greetings from the inner space!");
            }
        }
    }
}
```

```
public class Test {
    static void Main(string[] args) {
        FirstSpace.First.Greet();
        SecondSpace.Second.Greet();

        Outer.OuterClass.Greet();
        Outer.Inner.InnerClass.Greet();
    }
}
```

vagy pedig:

```
namespace Outer {
    public class OuterClass {
        public static void Greet() {
            Console.WriteLine("Greetings from outer space! 🚀");
        }
    }
}
namespace Outer.Inner {
    public class InnerClass {
        public static void Greet() {
            Console.WriteLine("Greetings from the inner space!");
        }
    }
}
```

using

```
using System;       // System library; pl: Console, WriteLine()
using FirstSpace;   // FirstSpace.First.Greet(); ⇒ First.Greet();
using SecondSpace;  // SecondSpace.Second.Greet(); ⇒ Second.Greet();

namespace FirstSpace {
    public class First {
        public static void Greet() {
            Console.WriteLine("Greetings from the first space!");
        }
    }
}

namespace SecondSpace {
    public class Second {
        public static void Greet() {
            Console.WriteLine("Greetings from the second space!");
        }
    }
}

class Test {
    static void Main(string[] args) {
        First.Greet();
        Second.Greet();
```

```
    }
}
```

Felsoroló típus (Enum)

- Speciális *class*, egy csoport konstans reprezentálására
- Szemléletesek és *típusbiztosak* (**type safe**, nem lehet másik classokkal keverni)

```
public enum PageState { empty, full }
```

Kivételek (Exceptions) és Kivételkezelés (Exception Handling)

```
public class MyException() : Exception { }
```

```
try {
    // do things...
} catch (MyException ex) {
    // catch the exception
} catch (MyOtherException ex) {
    // catch the other exception
} finally {
    // do something when control leaves try statement
}
```

Konvenciók és rövidítések

Privát adattagok ⟹ `_` és **camelCase**

```
private int _age;
```

Metódusnevek, *Classok*, *Namespacek*, *Tulajdonságok*, ... ⟹ **PascalCase**

```
public void FillPage() { ... }
```

Lokális változók ⟹ **camelCase**

```
int currentMaximum = ...
```

★ Ezeket természetesen nem kötelező, de ajánlott betartani (olvashatóság, egyértelműség)

new rövidítés:

```
Notebook notebook = new Notebook(32, NotebookType.lined);
```

⟺

```
Notebook notebook = new(32, NotebookType.lined);
```

return rövidítés:

```
public int Add(int a, int b) {
    return a + b;
}
```

⟺

```
public int Add(int a, int b) => a + b;
```

getter rövidítés:

```
public double R { get { return _r; } }
```

⟺

```
public double R => _r;
```

null (? és !)

- null : egy referencia ami nem mutat semmilyen objektumra
- alapból minden változó *non-nullable*
- ? operátor: nullabilitás változtatása *non-nullable*-ről *nullable*-re
- ! operátor: nullabilitás változtatása *nullable*-ről *non-nullable*-re
    - megmondjuk a compilernek, hogy ne aggódjon ebben az esetben a *null-biztonság* miatt

```
string line = "";    // lehetséges
string line = null;  // non-nullable változó nem lehet null

string? line = "";   // lehetséges
string? line = null; // lehetséges

string a = null!; // referencia most null, de eddig oké

string b = a.ToLower(); // itt a gond amikor dereferáljuk
(System.NullReferenceException)
```

Fileból olvasás

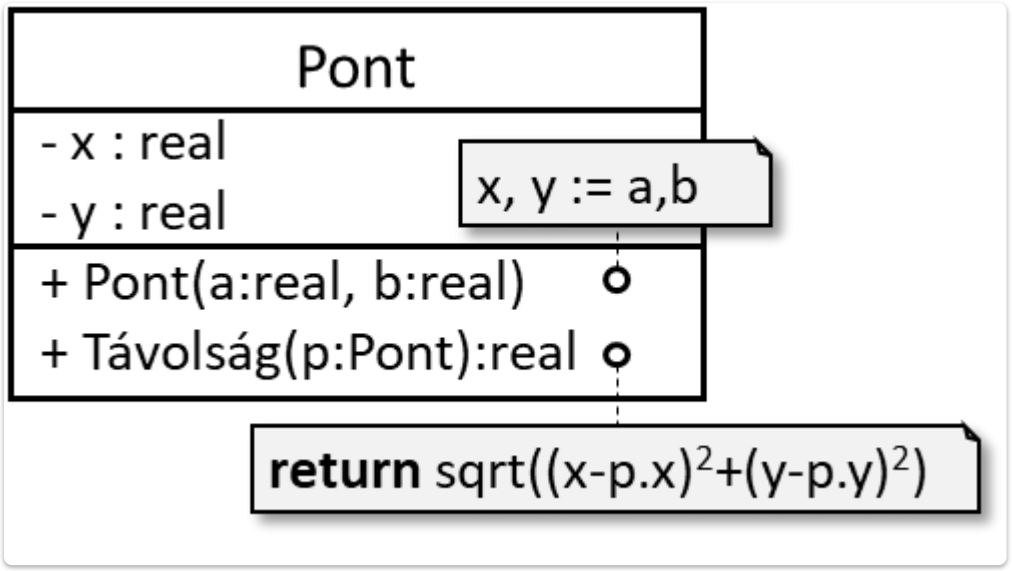
példa az összes sor beolvasására, és sorok kiírására a consolera

```
using (StreamReader reader = new StreamReader("file.txt")) {
    string? line = "";
    while ((line = reader.ReadLine()) != null) {
        Console.WriteLine(line);
    }
}
```

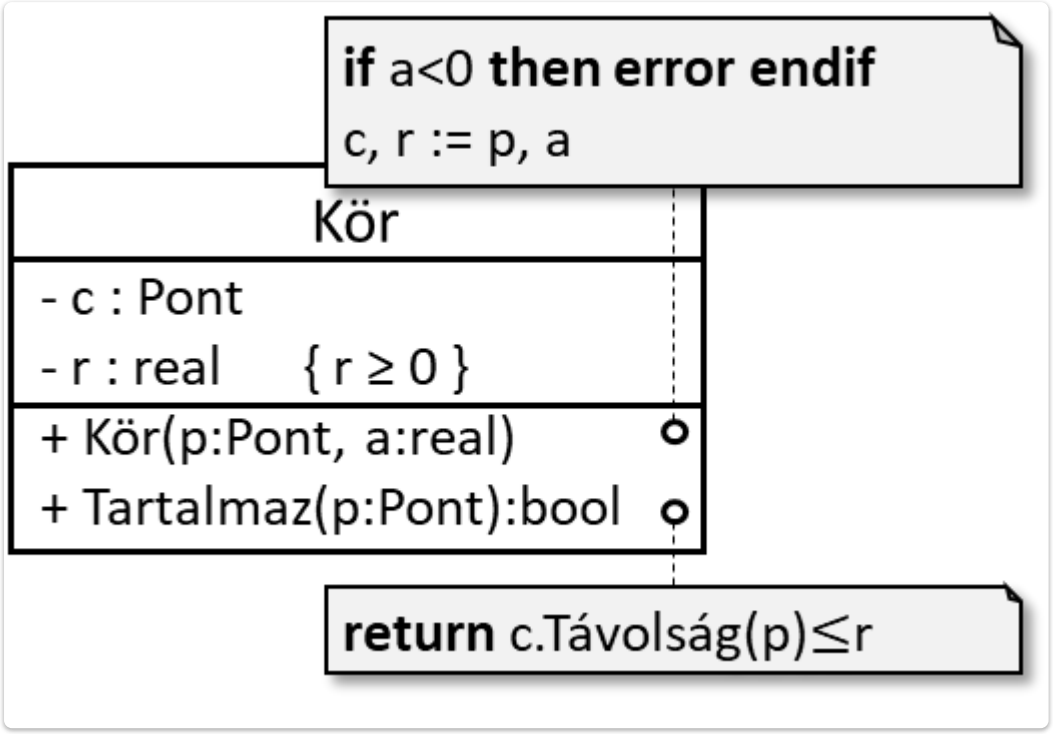
Órai Feladatok

Pont és kör

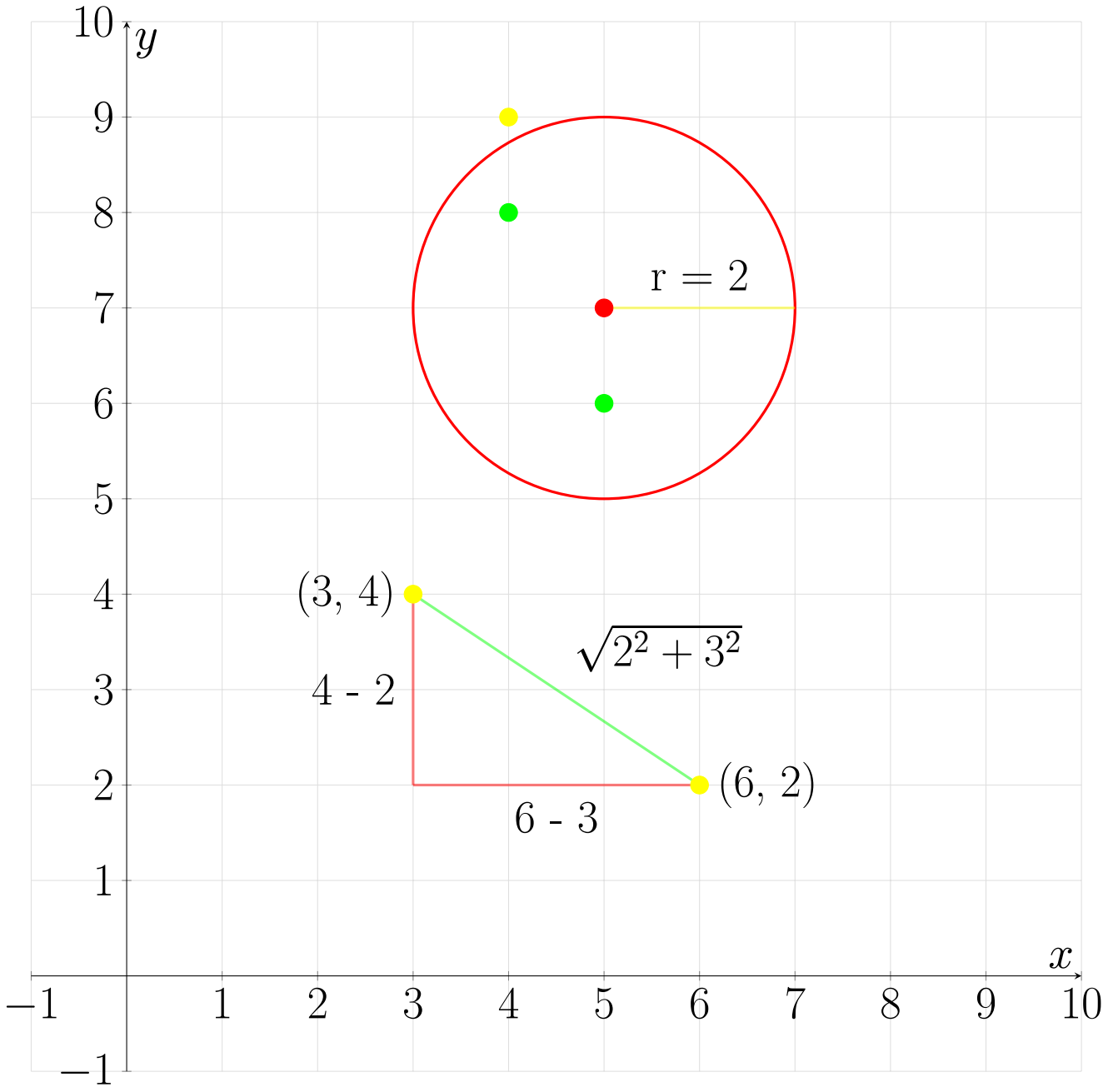
Pont



Kör



Illusztráció



Kisbeadandó (1/10) (1. batch)

★ Folyékony szappanadagoló