Interface

Interface-ek azok tulajdonképpen szabályok. Igazából ilyen szerződésnek tekinthetők az egyszerű interface-ek. Hogyha valamelyik osztály megvalósít egy interface-t, az azt jelenti, hogy az interface-ben lefektetett szabályokat ő tartalmazza, tehát megcsinálja azt a műveletsort, ami az interface-ben le van írva. Ennek következtében az interface-ek azok igazából publikus láthatóságúak alapértelmezetten, tehát nem is fogunk neki láthatóságot adni. Ennél jóval bonyolultabbak dolgokat is tudnak majd az interface-ek. De egyelőre mi most megelégszünk az egyszerű interface használattal, és majd amikor az objektumorientáltság haladóbb szintjeit vesszük, öröklődést meg virtuális metódusokat, akkor majd fogjuk ezt bővíteni. Először is adjunk hozzá egy interface-t. Osztállyal próbálom hozzáadni, és akkor onnan kiválasztom, hogy interface C#-ban az interface-eket nagy I-vel illik kezdeni. Ettől el lehet egyébként térni, de általában ezt fogja mutatni, hogy ő egy interface-. A környezetek egyébként különböző szint is adnak neki.

Amint létrejön az interface, internál láthatóságot rendelt hozzá, ugye, hogy a projekten belül legyen elérhető. Ezt hagyhatjuk így is, hogyha leszeded akkor automatikusan publikus a láthatósága és elérhető mindenhol. Mit fogunk belerakni az interface-be? Olyan tulajdonságokat és metódusokat, amiket elvárunk, hogy az az osztály, ami szeretné ezt a szabályrendszert teljesíteni, az abban az osztályban elérhető legyen ez a tulajdonság vagy metódus. Én mondjuk az ITerület-től azt fogom elvárni, hogy legyen egy double visszatérési értékű számított tulajdonsága. Tehát csak a get-et fogom megkövetelni, tehát egy számított tulajdonságba adja vissza bármilyen osztálynak, aminek van területe, a területtulajdonságot. Visszatérési érték, elnevezés és megmondtam, hogy lekérdezhető. Meg lehet adni azt is az interface-ben, hogy beállítható legyen, csak nem szokás és nem illik. Az osztály hadd döntse el saját maga, hogy ezt a tulajdonságot, hogyha megvalósítja, akkor a konstruktor állítja be, számított tulajdonság lesz és csak get-es része, vagy private set-es, azaz az osztályon belüli metódusok írhatják az adatrészt, vagy sima set-es, azaz kívülről is elérhetjük.

Lehetne itt kötelező metódust is megadni. Most egy nagyon-nagyon egyszerű megvalósítást fogunk nézni az interface-re.

Még egy olyan dolgot hozzárakunk ehhez a projekthez, amit most már érdemes megcsinálni, méghozzá írunk egy saját **kivételosztályt**. Nagyon egyszerű a saját kivételosztály írása. Ugye RosszMeretExceptiont fogok használni. Tehát hogyha mondjuk telkekkel fogunk most számolni a példaprogramban, egy teleknek a szélessége vagy a hosszúsága nulla vagy negatív szám, akkor váltsunk ki kivételt, hogy ezt azért nem nagyon hisszük el. Igazából többféleképpen használható a saját kivétel, vagy így simán exception-ből származtatva, és akkor ide idézőjelbe megadjuk azt a szöveget, amit fel szeretnénk dobatni vele. Csinálunk egy konstruktort neki és :base kulcsszóval megadjuk azt a szöveget, amit szeretnék kiíratni. Ugye minden kivétel az exception-ből származik, vagy akár ide paramétert is rakhatom. Most én azt fogom választani, hogy ide rakok egy paramétert, mert vissza szeretném adni, hogy melyik mező. És ha nem állítom be a mezőt, akkor annak legyen nulla az értéke, az alapértelmezett értéke. Tehát, hogy melyik mezőnek rossz a beállított értéke. De csinálhatok olyat is, hogy itt simán üres a zárójel, és akkor nem hivatkozok a kivételosztálynak a paraméterére. Ennyi igazából a saját exception osztály, tehát hogy adunk neki egy nevet, származtatjuk az exception-ből és írunk neki egy konstruktort, :base-zel pedig megadjuk azt a szöveget. A belsejében nincs is semmi.

Készítsünk egy Telek osztályt, mert igazából ez lesz az egyik, amin látszódni fog az, hogy mire is jó ez az interface. Egy osztálynak csak egy ősosztálya lehet, viszont akárhány interface-t megvalósíthat. És egy kettősponttal fogjuk megadni, hogy milyen interface-t kell megvalósítania. Vesszővel tudod fölsorolni az interface-eket. Én most azt fogom mondani, hogy ő az ITerulet interface-t legyen szíves megvalósítani, és alá is húzza egyből, és itt jelzi egyből, hogy nem implementáltuk az ITerulet Terület metódusát. Ez a problémája. Nagyon gyorsan tudunk rajta segíteni. Jobb egérgomb implement interface és egyből be is fogja dobni a Terulet metódust. Természetesen egy kivételt fog feldobni, hogy nem implementáltuk. Tehát hogyha megpróbálnánk erre a telekre a területet meghívni, akkor ez a kivé kiváltódna, merthogy nem írtuk meg a kódját, de legalább nem nekünk kell gondolkodni, hogyha az ITeruletet meg akarjuk valósítani, akkor milyen tulajdonságok, meg metódusok vannak benne, hanem így segít a környezet.

Még egy interface-t fogok most itt mutatni, mégpedig azért mert nagyon sok olyan interface van beépít a környezetbe, amit jól tudunk használni, és ebből az egyik leggyakoribb, amit szeretünk használni, ez az IComparable interface. Annyit fog tudni, ugye generikus interface, hogy összehasonlíthatóvá teszi az osztályunkat önmagával. Igazából ComperTo metódust fog csinálni, és ilyenkor visszaad egy 0 vagy egy -1, 1 értéket, hogyha két telket összehasonlítok. Hát most a területével fogjuk egyébként összehasonlítani. Ezt is implementáltuk és akkor ComperTo-val be is rakta egyébként azt a metódust, amit majd meg kell valósítanunk. Ugye ez a ComperTo, ez úgy fog működni, hogyha valamire meghívom a ComperTo műveletet és ugye zárójelbe be kell rakni, hogy mivel akarom össze összehasonlítani. Ha az első (amire hívom) a kisebb akkor -1, ha a második, tehát a paraméter a kisebb akkor +1, és ha egyezőek akkor pedig 0.

Ugye ez a Telek? azt jelenti, hogy other, amit kapunk paraméterről, az lehet nulla érték is. Tehát ez gyorsan ki fogom fejteni, mégpedig úgy, hogyha az other= null nem fogok új kivételt dobni, hanem ha nullértékkel hasonlítom össze, akkor visszatérek eggyel. Ez az alapértelmezett szokás, különben pedig a terület tulajdonságot ennek a példánynak összehasonlítom a paraméterről kapott területtulajdonsággal és ComperTo-val. És akkor ez majd megoldja, hogy milyen sorrendben vannak ők. Jó, tehát ez a ComperTo.

Na, azért kellene még néhány tulajdonság meg egyéb amivel tudnánk számolni, merthogy itt viszonylag kevés dolgot valósítottunk még meg a telekosztályból. Ugye lesz neki egy szélessége és egy hosszúsága és ezek full propertik lesznek rá. Ráadásul double-ösök, mert ugye nem fogjuk bármilyen értéket engedni, hogy beállítsanak, akkor engedünk egy telek szélességét átállítani, ugye lehet belőle adni részterületet is, tehát ezért engedem, hogy a teleknek a szélessége megváltozzon kívülről. De mondjuk nem lehet nulla, vagy nullánál kisebb a szélessége. Ha a szélesség értéke az nagyobb, mint nulla, akkor egyszerűen csak beállítjuk. Különben pedig ha már írtuk, akkor feldobjuk a RosszMeretExceptiont. Mégpedig úgy, hogy megmondom azt, hogy a szélesség adattulajdonság az, ami rossz, és adja vissza az értéket. Ugyanezt eljátsszuk a hosszúságra is. figyelve arra, hogy minden szükséges helyen ez át legyen írva. Ezen kívül én most fel fogok venni egy statikus számlálót. Én a statikust itt elől szeretem definiálni, de egy IdSzámlálót, ez egyszerűen csak azért, mert most a demrogramomnál én most generálni fogok 10 darab telket, és sorszámozom, tehát adok neki egy ID-t mindegyikhez, ami lekérdezhető lesz, az ID számláló fogja tárolni, hogy hányadik telek lesz generálva, és akkor így fogom majd logolni, meg kiírni a különböző telkeknek a területét véletlenszerűen fogjuk generálni.

És most már, hogy van szélességem meg hosszúságom, most már illik kiszámolni a területet. Tehát figyeljünk arra, hogy a nem implementált dolgokat nem hagyjuk bent. Úgyhogy most már van területem is.

És akkor annyit fogok még csinálni, hogy írunk egy konstruktort, mégpedig úgy, hogy kapja meg a telek szélességét, meg a telek hosszúságát. Ugye ez az ID számláló, ez pont arra lesz jó nekem, hogy új telek készül, akkor az ID számlálót azt léptetem. Ezzel megkapom az ID-t. Ezt egyébként össze is tudnám vonni, csak lehet, hogy kevésbé olvasható, hogy tehát ez a plusz plus ID számláló ide és akkor rövidebb kód lenne. És akkor beállítom a szélességtulajdonságnak a paraméterről kapott szélességet. Ugye ő majd elintézi, megdobja a megfelelő kivételt, hogyha nem jó a szélességadat, meg a hosszúságot.

És még egy dolgot fogok csinálni. A logolás meg az adatok kiírására csinálunk egy ToStringet, mert az nekünk jó. Kiírjuk a telek ID-ját, meg a szélességét, meg a hosszúságát és a területét.

Na ezzel gyakorlatilag készen lesz a telek osztálya. Mehetek vissza a program.cs-t macerálni. Ezt most itt kikomentezem, merthogy erre nem lesz szükségem. Ugye ezt csak azért raktam be, hogy honnan tudjuk kideríteni, hogy mit ad vissza a ComperTo, ha nem akarunk dokumentációt olvasni. És akkor annyit fogunk csinálni, hogy fölveszünk egy telektípusú listát. Ugye az a problémája, hogy nem látja ezt az osztályt merthogy a namespaceét be kell rakni. Úgyhogy ide be is raktam, ugye az nem alapértelmezett. És ezen kívül csinálok egy hibalistát. Hát nem teljesen ezt szerettem volna, hanem mondjuk azt mondani, hogy legyen 10. Próbáljunk meg generálni 10 darab telket. Legyen a szélesség az -3 és 10 között, hogy legyen lehetőségünk hibázni. Ugyanez legyen a hosszúság is.

És ugye hogyan kezeljük a kivételt, megpróbáljuk, try-catch blockba rakjuk, hogy próbáld meg megcsinálni a következő dolgot. Ugye a telkekhez adjunk hozzá egy új telket. Ugye kivételt dobhat a konstruktor, mert ha nem jó adatok, akkor a tulajdonság beállításánál kiváltódik a kivétel. Ha sikerült, ugye így néz ki a try-catch block, sikerült, akkor boldogak vagyunk, akkor bele fog kerülni a telkek listába. Ha nem sikerült, akkor pedig jön a kivételkezés. Ha már írtunk saját kivételt, akkor mindig a speciális kivétel osztályt próbáljuk el először elkapni. Ha sikerült elkapni, akkor egyszerűen a hibához add-dal hozzáadjuk (Ma nem igazán tudok gépelni) a hibaüzenetnek a message részét, és tulajdonképpen ez fog minket tájékoztatni. Ha nem csak ilyen RosszMeretException váltódhat ki, akkor lehet egyéb exceptiont is elkapni. Ide beírhatom azt, hogy exception vagy simán ezt hagyhatom catchel, hogyha ha nem akarom az exception szöveget kiváltani vagy fölhasználni. Ez itt valami váratlan hiba. Fogalmunk sincs, hogy mi történt, de valaki valamit elszúrt. Azért azt várjuk, hogy erre nem fogunk ráfutni egyébként.

Hogy miért jó az, hogyha megírjuk a ComperTo-t? A listák sorozatának van egy Sort metódusa, ami egyébként nem lambdázható, de amennyiben van az osztályhoz ComperTo művelet definiálva, akkor ez alapértelmezetten rendez. Szóval ettől mi boldogok leszünk. És akkor végig megyünk a telkek listán, és kiírjuk majd az összes teleket, meg hát kiírjuk a hibalistát is. Ugye végig megyünk a hiba listán, ha jól emlékszem, úgy hívtam, hogy hiba, és kiírjuk a hibákat. Elvileg ezzel van is egy működőképes programunk.

Még egy dologra szeretnék majd visszatérni, ugye a terület alapján rendezte őket nagyság sorrendben, merthogy az volt a ComperTo -nak az összehasonlítás művelete. És hát látszik,

hogy ténylegesen úgy is lehet ugye azért kerültek ide az ID-k, hogy látszó, hogy ez nem a véletlen műve. És látszik, hogy sikerült elkapni a megfelelő hibás értékeket.

Hogy miért jó igazából ez az interface. Azért, merthogy bármi helyett, tehát bármilyen osztály helyett használhatok interface-t, az az adott osztály megvalósítja az adott interface-t. Tehát létrehozhatok például egy olyan listát, aminek ITerület interface az elemtípusa, de akár csinálhatnánk simán ITerület típusú változót is. és kaphat ő értéket olyan osztályból, ami megvalósítja ezt az interface-t. Tehát így fog tudni a diák közös őst hozzárendelni, igazából egy interface-t a mondjuk a termékekhez és egy a listába, egy kollekcióba tárolni. Ezt is meg akartam mutatni, hogy területek.add és ha valamit hozzá lehet adni, és nem fog miatta sírni, hogy ez ténylegesen egy telek, tehát hogyha kiíratom a területek nulladik elemét, akkor ő ki is fogja írni, hogy ez a 10 x 20-as telek lesz, ami létre is jött.