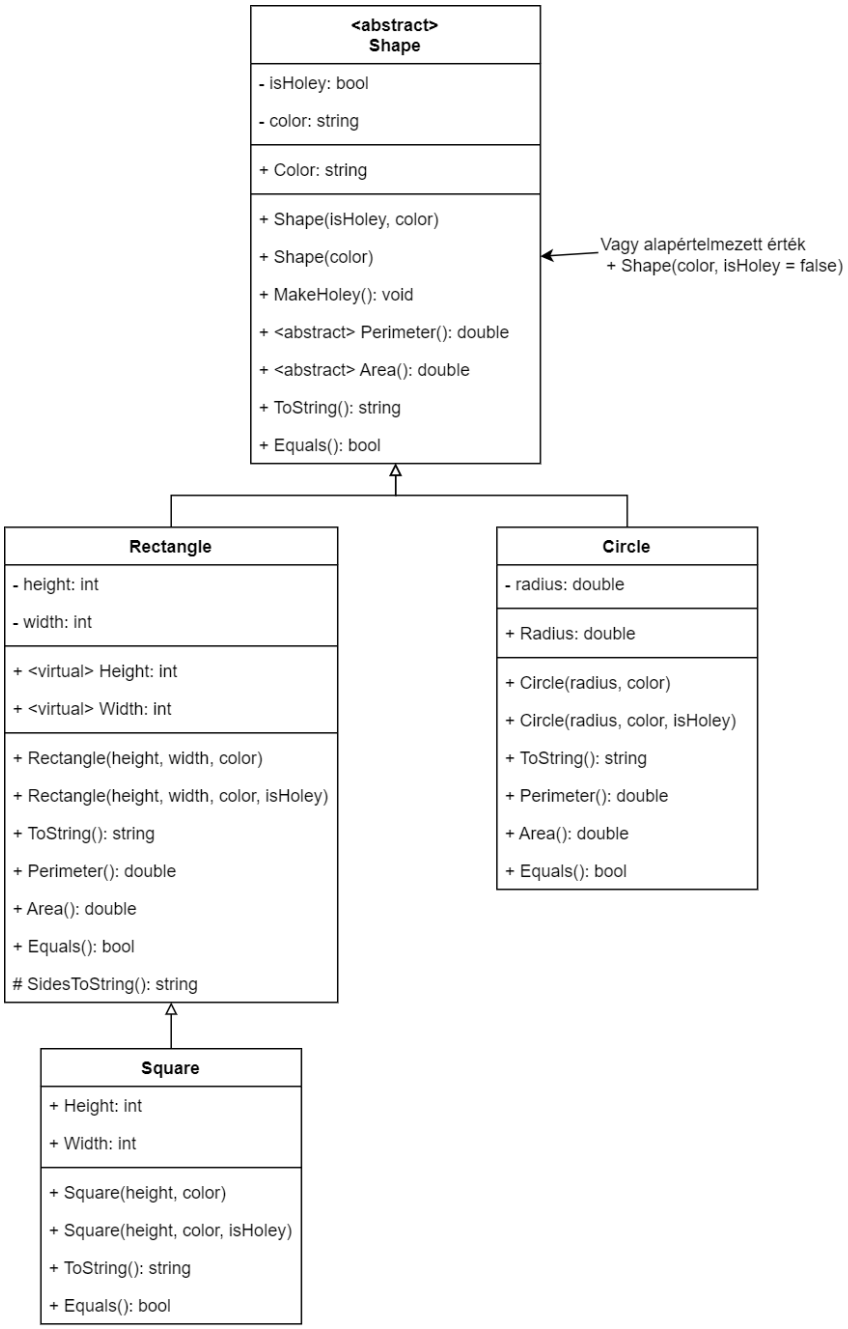
2. heti feladatok

1. órai feladat Valósítsa meg a következő osztályhierarchiát. Ügyeljen arra, hogy melyik metódusnak kell virtuálisnak lennie annak érdekében, hogy polimorf működést tegyen lehetővé.

- Ősosztályként legyen egy *absztrakt* **Shape**[1] osztályunk.
  - Tárolja el adattagként, hogy az adott **Shape** példány lyukas-e (`isHoley`), illetve mi a színe (`color`).
  - A szín legyen lekérdezhető és módosítható publikusan is.
  - Az osztály konstruktorán keresztül lehessen megadni a színt és a lyukasságot is. (Ha csak egy paraméterrel hívjuk meg a konstruktort, akkor csak a színt állítsa be, ilyenkor ne legyen lyukas a létrejövő **Shape** példány.)
  - Legyen az alakzatnak egy `MakeHoley()` metódusa, amit meghívva váljon lyukassá az alakzat.
  - Definiáljon egy `Perimeter`[2]`()` és egy `Area`[3]`()` metódust, de ezek ne legyenek implementálva.
  - Írja felül az **object** ősosztály `ToString()` metódusát úgy, hogy az adja vissza az objektum színét, lyukasságát, illetve a kerület és terület értékeket.
- A **Shape** osztályból származtasson le egy **Rectangle**[4] osztályt.
  - Adattagként tárolja el a téglalap magasságát (`height`) és szélességét (`width`).
  - A magasság és a szélesség is legyen lekérdezhető és módosítható publikus tulajdonságon keresztül.
  - Az osztály konstruktorán keresztül lehessen magasságot és szélességet is megadni, illetve ha szükséges, akkor más adatokat is.
  - A `ToString()` metódust írja úgy felül, hogy használja a **Shape** ősosztályban már implementált `ToString()` metódust, de többlet információként adja vissza, hogy téglalapunk van.
  - Mivel a **Rectangle** osztály példányosítható, implementálja az ősosztályban előírt, de ott nem implementált metódusokat.
- A **Rectangle** osztályból származtasson le egy **Square**[5] osztályt.
  - Implementálja úgy az osztályt, hogy az ősből megörökölt magasság és szélesség soha ne lehessen különböző.
  - Definiálja felül a **Rectangle** osztály `ToString()` metódusát olyan módon, hogy felhasználja az ős által szolgáltatott string értéket, ami elé a "Négyzet. " szöveget illeszti és kimenetként az újonnan összeállított stringet adja vissza.
- A **Shape** osztályból származtassa le **Circle**[6] osztályt is.
  - Egy körnek sugara[7] van, ennek megfelelően vegyen fel adattagot és publikus tulajdonságot, illetve implementáljon konstruktort.
  - A **Rectangle** osztályhoz hasonlóan implementálja az egyes örökölt metódusokat, figyelembe véve, hogy körrel kell dolgoznia.

[1] síkidom
[2] kerület
[3] terület
[4] téglalap
[5] négyzet
[6] kör
[7] radius

- Mindegyik osztály esetén írja felül az **object** ősosztályból örökölt `Equals` metódust.
- A programban hajtsa végre az alábbi feladatokat:
  - Tároljon el öt darab síkidomot egy tömbben.
  - Készítsen egy metódust, ami egy síkidomot kilyukaszt, ha annak nagyobb a területe, mint a kerülete.
  - Készítsen egy metódust, ami megadott oldalhosszak alapján létrehoz egy **Rectangle** vagy egy **Square** objektumot.
  - Készítsen egy metódust, ami síkidomok tömbjéből megadja a legnagyobb területű elemet.



2. gyakorló feladat Készítse el az alábbi felsorolt osztályokat.

- **Owner**[8] osztály
  - Kívülről írható/olvasható formában tárolja el a tulajdonos nevét.
  - Biztonsági okokból ne lehessen belőle származtatni.
- **BankingService**[9] osztály
  - A konstruktorban lehessen megadni a tulajdonost, ez a későbbiekben csak olvasható legyen.
  - Ebből az osztályból ne lehessen közvetlenül példányosítani.
- **BankAccount**[10] osztály
  - Legyen a **BankingService** osztály leszármazottja.
  - Konstruktorában lehessen megadni a tulajdonost.
  - Kívülről csak olvasható formában tárolja el az aktuális egyenleget.
  - Egy `Deposit`[11]`(amount`[12]`)` metódussal lehessen növelni az egyenleget.
  - Legyen egy hasonló paraméterű, de nem implementált `Withdraw`[13]`(amount)` metódusa is, aminek a visszatérési értéke egy logikai érték.
- **CreditAccount**[14] osztály
  - Legyen a **BankAccount** osztály leszármazottja.
  - A konstruktorban lehessen megadni a tulajdonos mellett a hitelkeret összegét, a későbbiekben ez csak olvasható legyen.
  - Valósítsa meg úgy a `Withdraw(amount)` metódust, hogy csak a hitelkeret mértékéig engedjen negatív számla egyenleget. Ellenkező esetben ne csökkentse az egyenleget és hamis visszatérési értékkel jelezze, hogy nem sikerült a kivétel.
- **SavingsAccount**[15] osztály
  - Legyen a **BankAccount** osztály leszármazottja.
  - Kívülről írható/olvasható formában tárolja el a kamat mértékét.
  - Az osztály egy statikus mezőjében tárolja el az alapértelmezett kamatot. Egy új megtakarítási számla létrehozásakor ez legyen a kamat kezdőértéke.
  - A `Withdraw(amount)` metódus ne engedje 0 alá csökkenni az egyenleget, visszatérési értéke jelezze, hogy sikerült-e a kivét.
  - Legyen egy `AddInterest()`[16] metódusa, ami jóváírja az esedékes kamatot.
- **BankCard**[17] osztály
  - Legyen a **BankingService** osztály leszármazottja.

[8] tulajdonos
[9] banki szolgáltatás
[10] bankszámla
[11] befizet
[12] összeg
[13] kivesz
[14] hitelszámla
[15] megtakarítási számla
[16] kamat hozzáadás
[17] bankkártya

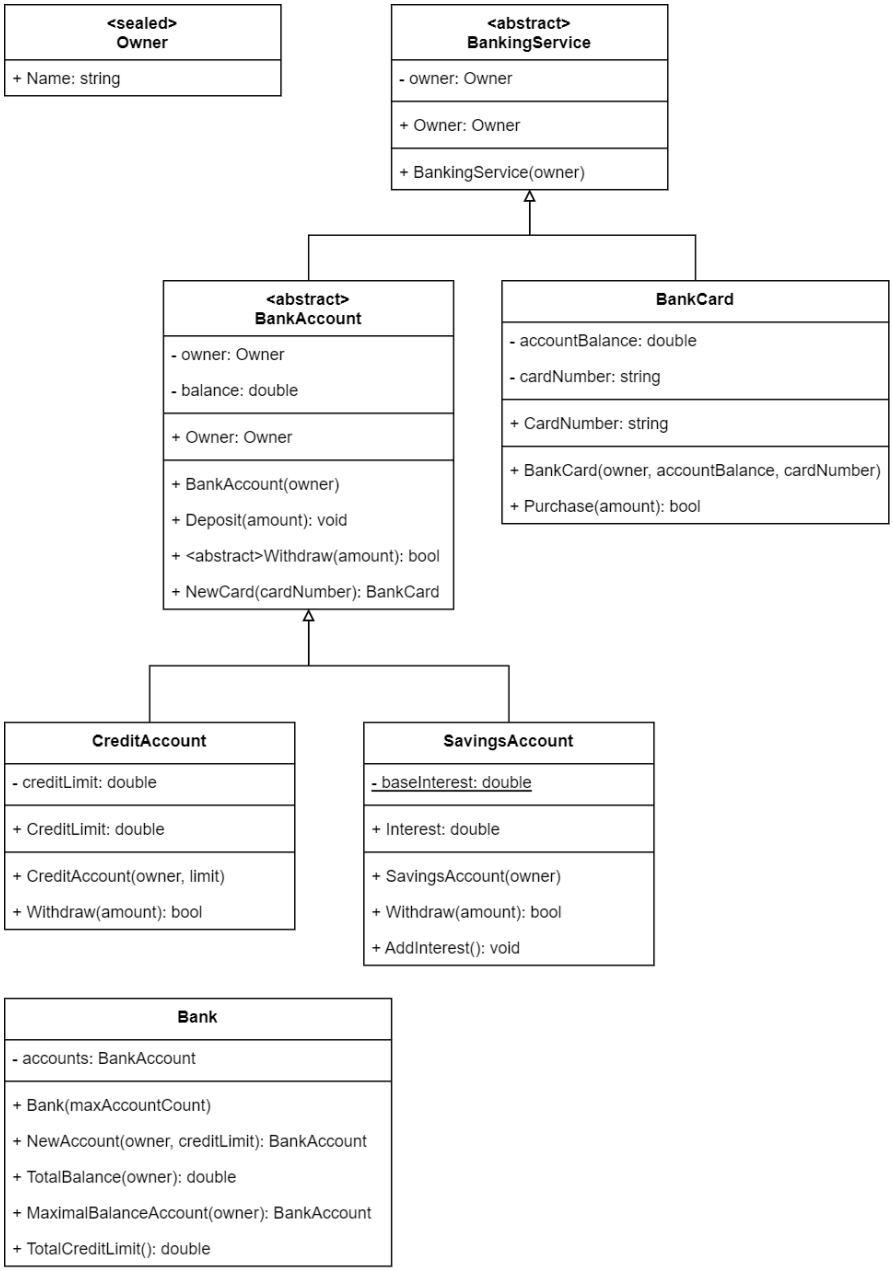
  - A konstruktorban lehessen megadni a tulajdonos mellett a hozzá tartozó mögöttes számlát, illetve a kártya számát.
  - A kártyaszám legyen kívülről olvasható, a mögöttes számla nem módosítható.
  - Készítsen egy `Purchase`[18]`(amount)` metódust, ami a paraméterként megadott összeggel megpróbálja csökkenteni a mögöttes számla egyenlegét, és visszatérési értéke legyen ennek sikeressége.

- **BankAccount** osztály kiegészítése
  - Egészítse ki a **BankAccount** osztály egy `NewCard(cardNumber`[19]`)` metódussal, amely a leendő kártyaszámot várja paraméterként.
  - A metódus hozzon létre egy új kártyát (az aktuális számlát és annak tulajdonosát adva meg a kártya adataiként) és legyen ez a metódus visszatérési értéke.

- **Bank** osztály
  - Tároljon el tetszőleges számú számlát, ezek maximális számát a **Bank** konstruktorában lehessen megadni.
  - Legyen egy `NewAccount(owner, creditLimit`[20]`)` metódusa, amelynek paraméterei egy **Owner** objektum és egy hitelkeret összeg. A hitelkeret összegének megfelelően hozzon létre hitel vagy megtakarítási számlát, ezt tárolja el, és ez legyen a metódus visszatérési értéke is.
  - Legyen egy `TotalBalance`[21]`(owner)` metódusa, amely visszaadja a paraméterként átadott tulajdonos számláinak összegyenlegét.
  - Legyen egy `MaximalBalanceAccount(owner)` metódusa, amely visszaadja a megadott tulajdonos legnagyobb egyenlegű számláját.
  - Legyen egy `TotalCreditLimit()` metódusa, amely visszaadja a bank által az összes ügyfélnek adott hitelkeretek összegét.

A fenti osztályok implementálását követően hozzon létre példa **Owner** és **Bank** objektumokat, majd próbálja ki a fenti funkciók működését.

[18] vásárlás
[19] kártyaszám
[20] hitelkeret
[21] összegyenleg


<sealed>
Owner
+ Name: string
<abstract>
BankingService
- owner: Owner
+ Owner: Owner
+ BankingService(owner)
<abstract>
BankAccount
- owner: Owner
- balance: double
+ Owner: Owner
+ BankAccount(owner)
+ Deposit(amount): void
+ <abstract>Withdraw(amount): bool
+ NewCard(cardNumber): BankCard
BankCard
- accountBalance: double
- cardNumber: string
+ CardNumber: string
+ BankCard(owner, accountBalance, cardNumber)
+ Purchase(amount): bool
CreditAccount
- creditLimit: double
+ CreditLimit: double
+ CreditAccount(owner, limit)
+ Withdraw(amount): bool
SavingsAccount
- baseInterest: double
+ Interest: double
+ SavingsAccount(owner)
+ Withdraw(amount): bool
+ AddInterest(): void
Bank
- accounts: BankAccount
+ Bank(maxAccountCount)
+ NewAccount(owner, creditLimit): BankAccount
+ TotalBalance(owner): double
+ MaximalBalanceAccount(owner): BankAccount
+ TotalCreditLimit(): double