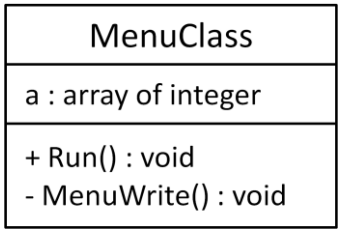
1. Gyakorlat

Készítsünk egy menüt egy egész számokat tartalmazó tömb műveleteinek kipróbálására!

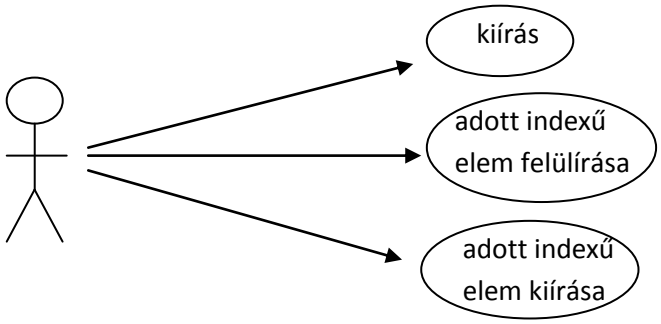
Legyen lehetőség a tömb összes elemének kiírására, a tömb *i*-dik elemének megadására illetve lekérdezésére (nem megfelelő *i* esetén kapjunk hibajelzést). Az egyes műveletekhez szükséges paramétereket a billentyűzetről olvassuk be, az eredményt a képernyőre írjuk ki.

Adjunk a feladatra objektum elvű megoldást! A menü típusát egy osztály írja le:



A megvalósításnál alkalmazzuk az alábbiakat!

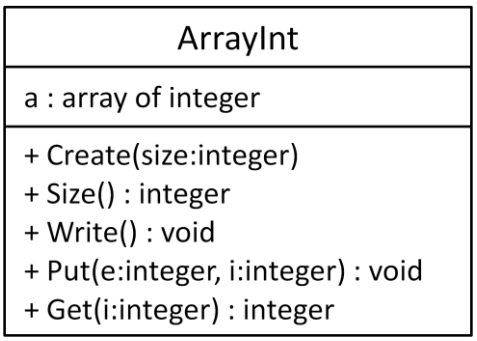
- Az „array of integer” típusú objektumot vector<int>-ként valósítsuk meg. Ennek méretét a MenuClass konstruktora állítsa be a konstruktor paramétereként megadott mérettel.
- A main() függvény feladata a MenuClass osztály példányosítása, azaz hozzon létre egy menü objektumot és hívja meg rá a Run() metódust.
- A Run() metódus valósítja meg a felhasználói eseteket:



Ennek érdekében a Run() metódus legyen egy do-while ciklus, amelynek ciklus magja kiírja a képernyőre a választható menüpontokat (MenuWrite()), beolvassa a felhasználó választását (hibás választ kezeljük le), majd egy switch elágazás a választól függően a megfelelő részfunkciót hajtsa végre.

2. Gyakorlat

Módosítsunk az 1. gyakorlat feladatának megoldásán úgy, hogy külön definiáljuk az egész számokat tartalmazó tömb típusát leíró osztályt, ahol az „array of integer” típusú objektumot dinamikusan lefoglalt tömbként valósítjuk meg (lásd előadás). Az alábbi osztálydiagram Create() metódusát konstruktorként implementáljuk: ez hozza létre a dinamikus tömböt. A destruktor gondoskodjon a felszabadításáról. Figyelem! Az ArrayInt osztály metódusai a Write() kivételével ne olvassanak a billentyűzetről és ne írjanak a konzolra!



Ekkor a MenuClass *a* adattagjának típusa ArrayInt lesz, és a Run() metódusban az ArrayInt metódusait kell meghívni.

További feladatok (az 1. a legfontosabb):

1. Egészítsük ki a menüt két újabb menüponttal:
   - Az egyikben adjuk értékül a tömböt egy itt létrehozottnak, majd végezzünk egymástól függetlenül különböző műveleteket mindkét tömbbön, és írjuk ki a tartalmukat.
   - A másikban hozzunk létre egy új tömböt a már meglevőből, majd végezzünk egymástól függetlenül különböző műveleteket mindkét tömbbön, és írjuk ki a tartalmukat.

   Javítsuk ki a programunk hibáját azzal, hogy megírjuk az ArrayInt osztály másoló konstruktorát és értékadás operátorát.
2. Az *i* index túlcsordulásának ellenőrzését a Put() és a Get() metódusok végezzék. Ezek hiba esetén dobjanak kivételt, amelyet a Run() metódusban kezeljünk le.
3. Tegyük lehetővé, hogy tetszőleges indextartományú tömbbel is lehessen dolgozni. A háttérben a tömb elemeit természetesen továbbra is egy dinamikusan lefoglalt 0-tól indexelhető tömbben tároljuk. Egy tömb létrehozásánál (Create()) tehát a méret helyett meg kell adni a tömb indextartományának alsó (lob) és felső (hib) határát. Ettől függ a tömb mérete: hib–lob+1, és az elemek módosítását illetve lekérdezését ennek megfelelő indexeléssel lehet végezni: a tömb i-dik eleme valójában a ténylegesen lefoglalt tömb i–lob-dik eleme lesz. Egészítsük ki az ArrayInt osztályt az indextartomány alsó és felső határát visszaadó metódusokkal, és tegyük lehetővé ezek kiírását a menü két újabb pontjának segítségével.
4. Használjunk operátor felüldefiniálást a metódusok megvalósításához!

   Indexelő operátor:

   ```
   int& operator[](int i);
   ```

   Kiíró operátor:

   ```
   friend std::ostream& operator<<(std::ostream &o, const Array &a);
   ```