Készítette: Varga Tamás János

Neptun Kód: EZUJIA

**Jegykiadás rendszer** ©

**Train osztály**

* Saját értekei vannak: (**string**)Name, (**int**)Space, (**LinkedList**)passengers, (**TakeOffAttendant**)Takeoff. Ezeknek értéket adunk a konstruktoron keresztül.
* A Train osztályba implementáljuk az **ICompareble**-t, majd felülírjuk az **CompareTo**-t, hiszen a feladat megoldásához szükséges a **Név** szerinti rendezés, valamint az optimalizációhoz is hasznát vesszük.
* Egy **TakeOffAttendant** nevű delegáltat hozunk létre ami egy Passenger és egy Train paramétert vár.
* Létrehozunk egy publikus **TakeOffAttendant** típusú **TakeOff** nevű eseményt melyet akkor sütünk el, ha az utasnak sikerült felszállnia az adott vonatra.
* **CompareTo** metódus, mely egy **object** paramétert vár; jelen esetben azt vizsgáljuk, hogy melyik vonat helye nagyobb a másikénál. **-1**-gyel tér vissza, ha az adott vonat szabad helyszáma **kisebb** **mint** amihez hasonlítjuk. **0**-val tér vissza, ha a két vizsgált vonatnak **ugyanannyi** a szabad helye és **1**-gyel tér vissza **ha nagyobb** a helyszáma mint a vizsgált vonaténak. Ezt a vonatok rendezésekor fogjuk használni.
* Létrehozunk egy **publikus** **Felszallit** nevű metódust, mely egy **Passenger** típusú értéket vár el. A metódus lényege, hogy meghívjuk a **Takeoff** eseményt mely egyik paramétere lesz az elvárt **passenger** és az adott vonat (**this**). Ezután az adott **passenger**-re meghívjuk az **Insert** metódust, ami a beszúrást végzi el és ezek után pedig a szabad helyek száma a szabad helyek és az utas helyfoglalásának a különbségét fogja eredményezni.

**Passenger osztály, ITicket interface**

* Az **ITicket** interfaceben létrehoztunk egy **TicketType** nevű enumot amiben a egyek fajtáit tároljuk el.(másodosztály, elsőosztály, biznisz). Ezen kívűl rendelkezik (**TicketType**)Type ,(**int**)Space,(**int**)Price nevű írható és olvasható (publikus) tulajdonságokkal, amelyet a **Passenger** osztály fog megvalósítani
* **ITicket** interfacet és az **ICompareble**-t implementáljuk, felruházzuk az osztályt plusz **publikus** tulajdonságokkal; (**string**)Destination és (**string**)Name, (**int**)Price.
* A konstruktorban bekérünk minden változót és értéket adunk nekik, kivéve a **Price**-nak ami a bekért **Destination(**uticél**)** hossza és a **Space**, vagyis a jegy helyfoglalásának a **szorzata**.
* Egy **switch-case** segítségével megadjuk, hogy melyik jegy (**tickettype**), hány helyet foglal.(másodosztály = 1; elsőosztály = 3; biznisz = 6)
* **CompareTo** metódusban azt kell vizsgálnunk, hogy melyik utas foglal több helyet, hiszen elsőbbséget élvez az aki több helyet foglal, vagyis az kerül fel előbb a vonatra akinek drágább a jegye.

**LinkedList, LinkedListElement osztályok**

* A láncoltlista osztályunkhoz kell készítenünk egy láncoltlista elemeket tartalmazó osztályt amelyben egy publikus **Passenger** típusú tartalmat és egy **LinkedListElement** típusú mutatót foglal magában. Mindkét tulajdonság **írható és olasható**.
* Létrehozunk egy utaskezelő delegáltat (**PassengerAttendant**) ami egy **Passenger** típusú értéket vár.
* Létrehozunk két eseményt(„**Ticket\_Buy**”, „**Delete\_Passenger**”) ami **PassengerAttendant** típusúak és akkor sütjük el ezeket az eseményeket, ha egy utas vásárolt jegyet(„**Ticket\_Buy**”) vagy éppen egy utast kitöröltünk(„**Delete\_Passenger**”) a listából.
* Létrehozunk két **publikus** metódust a két eseménynek (**TB\_Happened**, **PD\_Happened**) amikben „kiváltjuk” az eseményeket.
* Létrehozunk egy **Insert** nevű publikus metódust aminek a paramétere egy **Passenger** típusú érték. Ez lesz a **Beszúrás** metódusa a láncolt listánknak. Vizsgálandó esetek: Ne legyen **null**, és **név** szerint rendezetten szúrja be a listába az adott utast. A metódus végén meghívjuk a **TB\_Happened** eseményt
* Ha megegyezik az adott utas neve azzal az utassal akit már egyszer felvettünk, akkor kivételt dobunk, hogy az az utas már egyszer fel lett véve.(**PassengerExistsException**)
* Létrehozunk egy **Delete** nevű publikus metódust aminek a paramétere szintén egy **Passenger** típusú érték. Ez lesz a láncolt listának a **Törlés** metódusa. Vizsgálandó esetek: Ne legyen **null** értékű a lista és a keresett utas **GetHashCode**-ja egyezzen meg a listában szereplő utas **GetHashCode**-jával. A metódus végén meghívjuk a **PD\_Happened** nevű eseményt.
* Ha nem találtuk meg az adott személyt a listában akkor dobjunk kivételt ezesetben.(**CouldntFindException**)
* Létrehozunk egy **Bejaras** nevű metódust amelyben referencia értéktípusként bekérünk egy **Passenger** típusú beépített listát.
* Egy **ToArray** metódust hozunk létre ami egy **Passenger** típusú tömböt vár visszatérési értékként. Ebben a metódusban a **Passenger** listát, tömbbé konvertáljuk át.

**PassengerExistEception kivétel**

* **Exception** osztály leszármazottja. Felülírjuk a **Message**-et, a saját kivételünknek megfelelően. Ez a kivétel a **már létező** utas(okra) figyelmeztet.

**CouldntTakeOffException kivétel**

* **Exception** osztály leszármazottja. Egy üzenetet adunk át a **base**-nek, ha az utas már nem bírt felszállni a vonatra.

**CouldntFindException kivétel**

* **Exception** osztály leszármazottja. Ha nem találtuk meg a keresett személyt a **Delete** metódusban akkor egy hibaüzenetet ír ki, hogy nem találtuk meg.

**Tesztelés, Program.cs**

* Létrehozunk egy-egy név, utas, uticél, vonat generátor metódusokat ahol létrehozzuk és/vagy elkészítjük a különböző személyeket, uticélokat, stb…
* Létrehozunk egy-egy rendezési metódust; **TrainSort, PassengerSort**. Ez azért fontos, mert amikor az utasok ülőhelyét akarnánk kiosztani akkor először a legnagyobb „elemű” utas kerüljön leültetésre.
* **SY\_CouldntTakeOff**: A tömbbe, ahol az érték nem NULL arra kivételt dobunk, mert akkor az az utas már nem tudott felszállni a vonatra.
* **Trasportage** metódus: Végig iterálunk egy külső ciklussal ahány utas van. A while ciklus előtt létrehozunk egy **j** indexet amely a vonat indexelését szolgálja. Ha az **i**-edik utas szükséges férőhelyszáma kisebb vagy egyenlő mint a **j**-edik vonat szabad férőhelye akkor a **Felszallit** metódust meghívjuk a **j**-edik vonatra es bele tesszük az **i**-edik utast, ezután azt az utast **NULL**-ra állítjuk, vagyis **Ő** fel tudott szállni a vonatra. A külső ciklus végén meghívjuk a **TrainSort** rendezési metódust, ezáltal folyamatosan fognak frissülni csökkenő sorrendben a vonatok a férőhely számuk alapján. Ha már nem fér fel pl: az első vonatra az az adott utas akkor a **j** tovább lép egyel és ugyanúgy megvizsgálja.
* **Main()**: Tetszőlegesen létrehozunk pl.: egy 5 elemű **Train** típusú tömböt ahova a vonatok vannak eltárolva. A utasokat feliratkoztatjuk a vásárlási és a törlés eseményekre. Tetszőleges számú utasokat létrehozunk, majd teszteljük az **Instert** és a **Delete** metódust. Elhelyezzük az utasokat egy tömbbe, majd az utasokra meghívjuk a rendezési metódust. Ezután a **Transportage** metódus megfelelően kiosztja a helyeket a vonaton, elhelyezi az utasokat. **Sy\_CouldntTakeOff** metódust meghívuk azokra az utasokra akik nem tudtak felszállni a vonatra.

*Megjegyzés: A korlátozott számú nevek miatt előfordulhat, hogy ugyanazt a nevet generálja a program, ezért hibát fog dobni.(***PassengerExistEception***)*