PROGRAMOZÁS 1.

2. labor

Tulajdonság (Properties)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties

Generikus lista (List<T> Class)
https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8

Object.Equals metódus (Object.Equals Method)
https://docs.microsoft.com/en-us/dotnet/api/system.object.equals?view=netframework-4.8

Random osztály (Random Class)
https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8

Konstruktorok (Constructors)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors

Referencia mutató (this)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/this

DateTime struktúra (DateTime Struct)
https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=netframework-4.8

Osztály létrehozása (Declaring Classes)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes#declaring-classes

Objektum létrehozása (Creating objects)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes#creating-objects

snippetek (C# code snippets)
https://docs.microsoft.com/en-us/visualstudio/ide/visual-csharp-code-snippets?view=vs-2017

Készítette: Dr. Katona József, PhD
egyetemi docens, tanszékvezető
email: katonaj@uniduna.hu

A gyakorlati anyag letölthető:
https://dufoffice365-my.sharepoint.com/:f:/g/personal/katonaj_uniduna_hu/Eg5kJEIa_sRFlDacXMCuUzEB4vK8ClZBjp6TT1MO3P9U9A?e=Qgffmt

A példaprogramok letölthetők:
https://dufoffice365-my.sharepoint.com/:f:/g/personal/katonaj_uniduna_hu/EnSASoX17NRMshI0CPcizTwBY61cC50m20Y7zzDUo5Jk7g?e=fjjfh2

TARTALOMJEGYZÉK

1. PersonGeneratorApp: „Véletlen személy” 3
1.1. A feladat ismertetése 3
1.2. A megvalósítandó alkalmazás lehetséges UML tervezetei 3
1.2.1. Osztálydiagram 3
1.2.2. Objektumdiagram 4
1.3. A feladat megoldásának részletezése 4
1.4. A feladat megoldása 6
1.4.1. Person.cs 6
1.4.2. Address.cs 6
1.4.3. Car.cs 7
1.4.4. RandomPerson.cs 7
1.4.5. Program.cs 9
2. Gyakorló feladatok 10
2.1. A feladat ismertetése (CalculatorApp) 10
2.1.1. Osztálydiagram 10
2.2. A feladat ismertetése (TextApp) 10
2.2.1. Osztálydiagram 10
2.3. A feladat ismertetése (AirplaneApp) 10
2.3.1. Osztálydiagram 11
2.3.2. Objektumdiagram 12

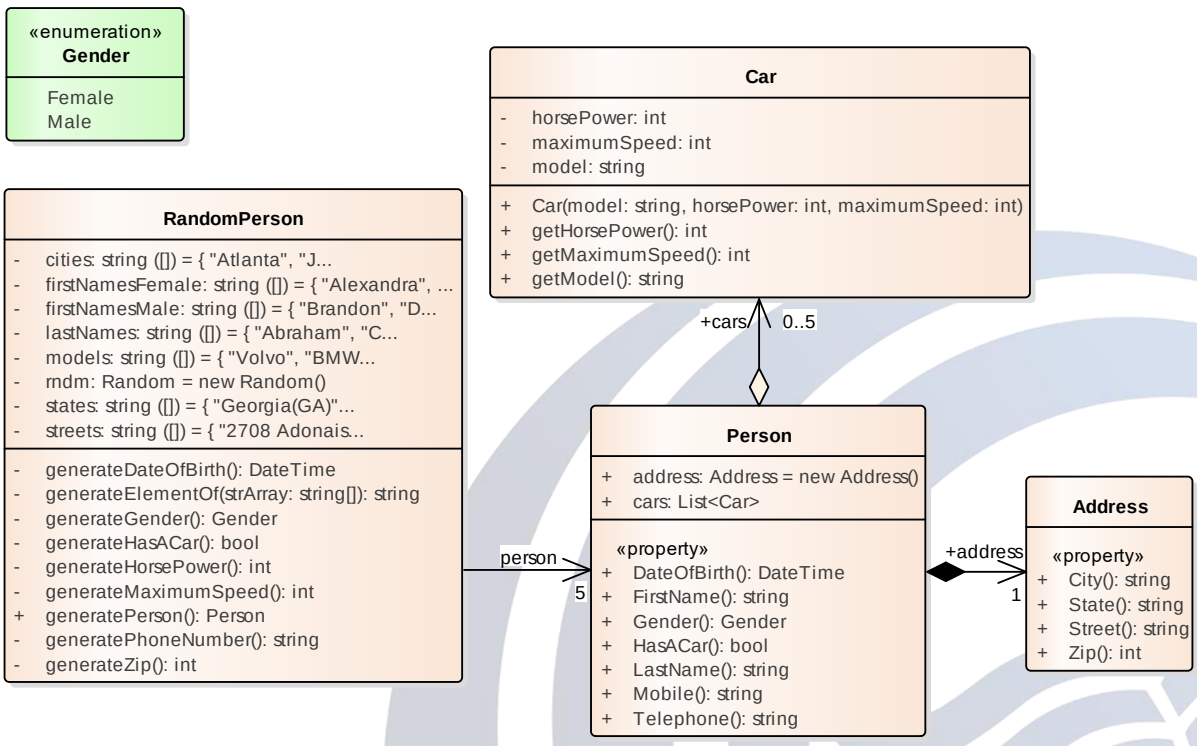
1. PersonGeneratorApp: „Véletlen személy”

1.1. A feladat ismertetése

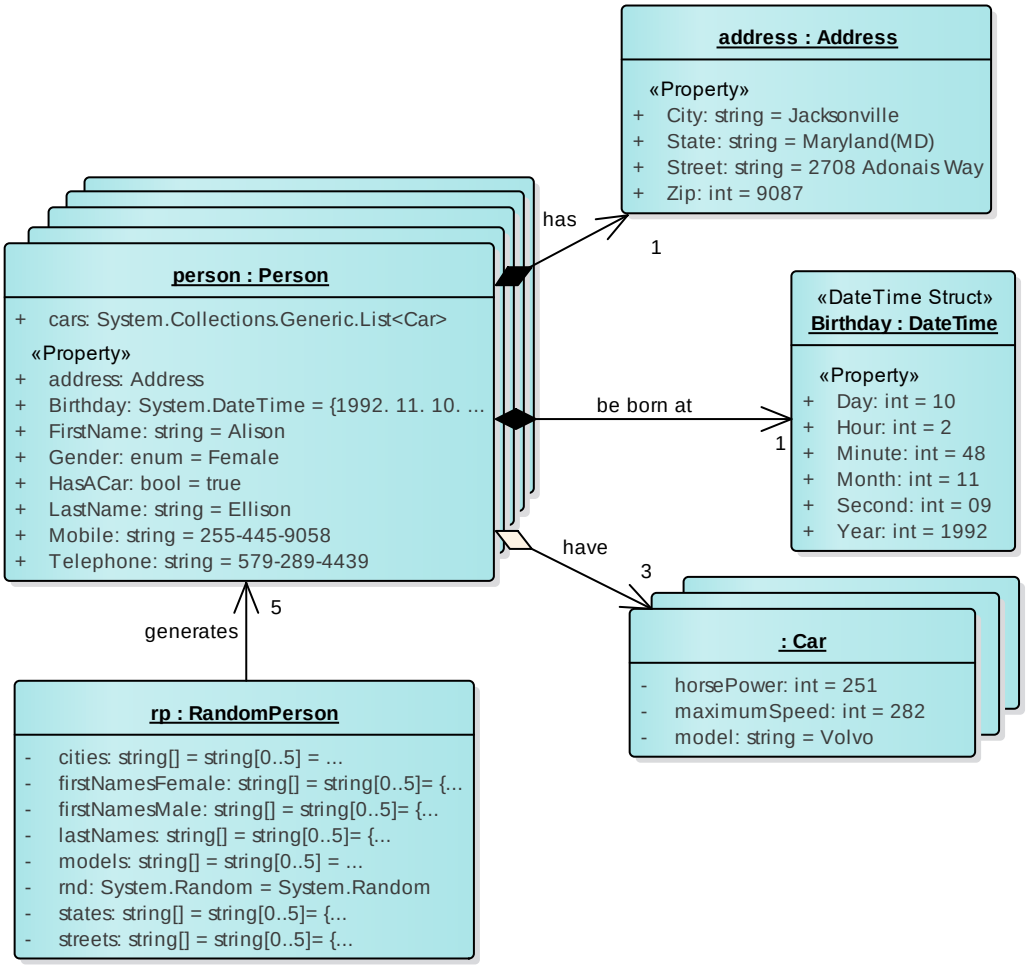
Tervezzen meg, majd készítsen el egy alkalmazást „véletlen” személy(ek) generálásához. Egy személyről a következő adatokat tárolja: vezetéknév, keresztnév, nem, születési dátum, telefonszám, mobilszám, lakcím, a birtokában lévő autó tulajdonságai. A lakcím esetében tárolja a várost, megyét, utcát és az irányítószámot, külön ügyeljen arra, hogy egy személyhez csak egy lakcím tartozhat. Egy személy autójának tulajdonság értékeit csak akkor tartsa nyilván, ha birtokol autót, arra viszont figyeljen, hogy egy személy akár több (maximum 5 darab) autót is birtokolhat. Amennyiben egy személynek van autója, akkor a következő tulajdonságokat tárolja: modellnév, lóerőszám és maximális sebesség. A személy vezetéknév, keresztnév, utca, város, megye/állam és autójának modell adatait véletlenszerűen válassza ki az adott adatok tárolására létrehozott tömbökből. Ügyeljen külön a női és férfi nevek generálására. Az irányítószám 1…9999 tartományba essen, a születési dátum 1900. 01. 01-től napjainkig tarthat, amíg a telefonszám és a mobil szám {100…999}-{100…999}-{1000…9999} formának és értéktartománynak megfelelő lehet. Annak eldöntése, hogy az adott személy birtokol-e autót a véletlen feladata. Egy autó lóerejének 115…300, amíg a maximális sebességének 180…300 tartományba kell esnie. Az osztály megvalósítását követően készítsen öt darab objektumpéldányt a személyek adatainak megjelenítésére.

1.2. A megvalósítandó alkalmazás lehetséges UML tervezetei

1.2.1. Osztálydiagram



1.2.2. Objektumdiagram



1.3. A feladat megoldásának részletezése

A terveket felhasználva elkezdhetjük a megvalósítást. Összesen négy osztályt készítettünk; *Address*, *Car*, *Person* és *RandomPerson*.

Az *Address* osztályban nyilvános láthatósági szint mellett, tulajdonságokban tároljuk valamely személy városának, megyéjének és utcájának nevét, valamint irányítószámát.

A *Car* osztályban magán láthatósági szint mellett, adatmezőkben tároljuk valamely autó modelljének nevét, lóerejének és maximális sebességének értékét. Az osztály a három adattagon felül további négy, nyilvános láthatósági szinttel rendelkező tagfüggvényt tartalmaz:

- **Car(string model, int horsePower, int maximumSpeed)**: Paraméteres konstruktor, amely a paraméterben kapott értékeket állítja be az osztály adatmezőinek értékeiként. A *this* mutató segítségével kiküszöböljük a takarás problémáját.
- **string getModel**: Adott objektumhoz tartozó modellnevet adja vissza *string* típusként.
- **string getHorsePower**: Adott objektumhoz tartozó lóerő értéket adja vissza *int* típusként.

- **string getMaximumSpeed**: Adott objektumhoz tartozó maximális sebesség értéket adja vissza *int* típusként.

A *Person* osztályban nyilvános láthatósági szint mellett, tulajdonságokban tároljuk valamely személy vezeték- és keresztnevét, nemét, születési dátumát, telefonszámát, mobilszámát, valamint azt, hogy birtokol-e autót(kat). Mindezek a tulajdonságértékek mellett kompozíciós kapcsolat mutatható ki a *Person* és az *Address* osztályok között, mivel ebben a speciális esetben egy személynek kötelezően rendelkeznie kell valamilyen lakcímmel. A *Person* és *Car* osztály között aggregációs reláció fedezhető fel. Egy autó pontosan egy személyhez tartozhat, azonban nem minden személynek van autója, továbbá egy személynek maximálisan öt autója lehet. Az autók tárolása egy generikus listában történik.

A *RandomPerson* osztály leírja milyen módon generáljuk az adatokat egy adott személyhez. Az osztály tartalmaz egy *Random* példányt, valamint hét darab *string* tömböt:

- **firstNamesFemale**: öt darab női keresztnevet tartalmazó tömb.
- **firstNamesMale**: öt darab férfi keresztnevet tartalmazó tömb.
- **lastNames**: öt darab vezetéknevet tartalmazó tömb.
- **streets**: öt darab utcanevet és házszámot tartalmazó tömb.
- **states:** öt darab államnevet tartalmazó tömb.
- **models:** öt darab autó modellt tartalmazó tömb.

Az *rnd* példányon, valamint a *string* tömbökön kívül az osztály további 9 metódust ír le:

- **string generateElementOf**: A paraméterül kapott tömbből véletlenszerűen kiválaszt egy értéket, majd visszaadja.
- **int generateZip**: 1 és 9999 között generál egy véltelenszámot, majd visszaadja azt.
- **Gender generateGender**: 0 és 1 közötti véletlenszám alapján eldönti, hogy Female-t vagy Male-t adjon vissza. (0=Female, 1=Male)
- **generateBirthDate**: Kiindulási időpontként 1900 01. 01. adjuk meg, majd meghatározunk egy tartomány, a kiindulási dátumtól napjainkig. Ezt követően a kiindulási dátumhoz hozzáadunk annyi napot, amennyit a véletlenszám generátor meghatároz.
- **string generatePhoneNumber**: {100…999}-{100…999}-{1000…9999} formának és korábban meghatározott értéktartománynak megfelelő számot ad vissza.
- **bool generateHasACar**: Amennyiben a generált érték 0 úgy igazzal, ellenben hamissal tér vissza.
- **int generateHorsePower**: 115 és 300 között generál egy véltelenszámot, majd visszaadja.
- **int generateMaximumSpeed**: 180 és 300 között generál egy véltelenszámot, majd visszaadja.

- **Person generatePerson(Person person)**: A függvény a *Person* osztály egy objektumát hozza létre, majd a *generateGender* metódust felhasználva inicializálja a példány *Gender* tulajdonságértékét, ezt követően attól függően, hogy a *Gender* milyen értéket vett fel, véletlenszerűen legenerálódik a megfelelő nemhez tartozó keresztnév és vezetéknév. Tovább folytatva ezt a metodikát beállítja a születési dátumot, a telefonszámot, valamint a mobilszámot is. A *person* példány *address* objektumát felhasználva az adott személyhez tartozó lakcím értéket is beállítjuk. A *generateHasACar* metódus eredményétől függően, amennyiben az igaz értéket vesz fel, úgy definiálunk egy *Car* típusú listát, majd véletlenszám generálás segítségével eldöntjük, hogy az adott személyhez hány autó tartozik. A mennyiség meghatározását követően feltöltjük a listát az autók osztály példányaihoz tartozó értékekkel. Végül visszaadjuk a *person* objektumot.

A *Program* osztályban található az alkalmazás belépési pontja, a *Main* függvény, ahol az osztályok példányát hozzuk létre. Az első példányunk a *RandomPerson* osztályhoz kapcsolódik, majd egy *for* cikluson belül létrehozunk öt darab személyt, akiket a korábban létrehozott *personGenerator* objektum *generatePerson* metódusának segítségével generáltunk. Végül az adatokat megjelenítjük a konzolon. Az autók tulajdonságát, végig haladva a *person* objektumhoz köthető autók listáján, csak akkor írja ki, ha azok léteznek, egyébként a „HAS NO ANY CARS” szöveget jelenítjük meg.

1.4. A feladat megoldása

1.4.1. Person.cs

```
using System;
using System.Collections.Generic;

namespace PersonGeneratorApp
{
    public enum Gender { Female, Male }
    public class Person
    {
        public Address address = new Address();
        public List<Car> cars;
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Gender Gender { get; set; }
        public DateTime DateOfBirth { get; set; }
        public string Telephone { get; set; }
        public string Mobile { get; set; }
        public bool HasACar
        {
            get { return cars != null && cars.Count > 0; }
        }
    }
}
```

1.4.2. Address.cs

```
namespace PersonGeneratorApp
{
```

```
    public class Address
    {
        //full-property
        //private string city;
        //public string City
        //{
        //    get { return city; }
        //    set { city = value; }
        //}
        public string City { get; set; }
        public string State { get; set; }
        public string Street { get; set; }
        public int Zip { get; set; }
    }
}
```

1.4.3. Car.cs

```
namespace PersonGeneratorApp
{
    public class Car
    {
        private string model;
        private int horsePower, maximumSpeed;

        public Car(string model, int horsePower, int maximumSpeed)
        {
            this.model = model;
            this.horsePower = horsePower;
            this.maximumSpeed = maximumSpeed;
        }
        public string getModel()
        {
            return model;
        }
        public int getHorsePower()
        {
            return horsePower;
        }
        public int getMaximumSpeed()
        {
            return maximumSpeed;
        }
    }
}
```

1.4.4. RandomPerson.cs

```
using System;
using System.Collections.Generic;

namespace PersonGeneratorApp
{
    public class RandomPerson
    {
        private Random rndm = new Random();
        private string[] firstNamesFemale = { "Alexandra", "Alison", "Maria",
"Sophie", "Wanda" },
            firstNamesMale = { "Brandon", "David", "Gordon", "Jonathan", "Peter" },
            lastNames = { "Abraham", "Campbell", "Elison", "Henderson", "Johnston" },
            streets = { "2708 Adonais Way", "4154 Cherry Tree Drive", "3466 Wilmar
Farm Road", "1949 Jadewood Drive", "501 Blane Street" },
```

```
            cities = { "Atlanta", "Jacksonville", "Lanham", "Wheatfield", "Fairview
Heights" },
            states = { "Georgia(GA)", "Florida(FL)", "Maryland(MD)", "Indiana(IN)",
"Missouri(MO)" },
            models = { "Volvo", "BMW", "Jaguar", "Audi", "Ford" };

        private string generateElementOf(string[] strArray)
        {
            return strArray[rndm.Next(strArray.Length)];
        }
        private int generateZip()
        {
            return rndm.Next(1, 10000);
        }
        private Gender generateGender()
        {
            return rndm.Next(2) == 0 ? Gender.Female : Gender.Male;
        }
        private DateTime generateDateOfBirth()
        {
            DateTime start = new DateTime(1900, 1, 1);
            int range = (DateTime.Today - start).Days;
            return
start.AddDays(rndm.Next(range)).AddHours(rndm.Next(24)).AddMinutes(rndm.Next(60)).AddS
econds(rndm.Next(60));
        }
        private string generatePhoneNumber()
        {
            return rndm.Next(100, 1000) + "-" + rndm.Next(100, 1000) + "-" +
rndm.Next(1000, 10000);
        }
        private bool generateHasACar()
        {
            return rndm.Next(2) == 0;
        }
        private int generateHorsePower()
        {
            return rndm.Next(115, 301);
        }
        private int generateMaximumSpeed()
        {
            return rndm.Next(180, 301);
        }
        public Person generatePerson()
        {
            Person person = new Person();

            person.Gender = generateGender();
            person.FirstName = generateElementOf(person.Gender.Equals(Gender.Female) ?
firstNamesFemale : firstNamesMale);
            person.LastName = generateElementOf(lastNames);
            person.DateOfBirth = generateDateOfBirth();
            person.Telephone = generatePhoneNumber();
            person.Mobile = generatePhoneNumber();
            person.address.City = generateElementOf(cities);
            person.address.State = generateElementOf(states);
            person.address.Street = generateElementOf(streets);
            person.address.Zip = generateZip();
            if (generateHasACar())
            {
                person.cars = new List<Car>();
                int howMany = rndm.Next(1, 5 + 1);
```

```
                for (int i = 0; i < howMany; ++i)
                    person.cars.Add(new Car(generateElementOf(models),
generateHorsePower(), generateMaximumSpeed()));
            }

            return person;
        }
    }
}
```

1.4.5. Program.cs

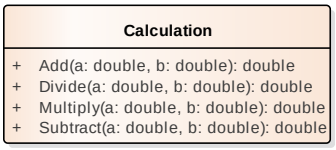
```
using System;

namespace PersonGeneratorApp
{
    public class Program
    {
        static void Main(string[] args)
        {
            const int N = 5;
            RandomPerson rp = new RandomPerson();
            for (int i = 0; i < N; ++i)
            {
                Person person = rp.generatePerson();
                Console.WriteLine("==================");
                Console.WriteLine("BASIC INFORMATION");
                Console.WriteLine($"Firstname: {person.FirstName}\n\rLastname:
{person.LastName}\n\rGender: {person.Gender}\n\rDate Of Birth: {person.DateOfBirth}");
                Console.WriteLine("\n\rCONTACT INFORMATION");
                Console.WriteLine($"Telephone: {person.Telephone} Mobile:
{person.Mobile}\n\rCity: {person.address.City}\n\rState:
{person.address.State}\n\rStreet: {person.address.Street}\n\rZip:
{person.address.Zip}");
                Console.WriteLine("\n\rCAR INFORMATION");
                if (person.HasACar)
                    foreach (Car car in person.cars)
                        Console.WriteLine($"Model: {car.getModel()}\n\rHorsePower:
{car.getHorsePower()}\n\rMaximum speed: {car.getMaximumSpeed()}");
                else
                    Console.WriteLine("HAS NO ANY CARS");
                Console.WriteLine("==================");
                Console.WriteLine();
            }
        }
    }
}
```

2. Gyakorló feladatok

2.1. A feladat ismertetése (CalculatorApp)

Tervezzen meg, majd implementáljon egy egyszerű osztályt, amelynek objektuma képes két valós számon a négy alapvető aritmetikai művelet végrehajtására.

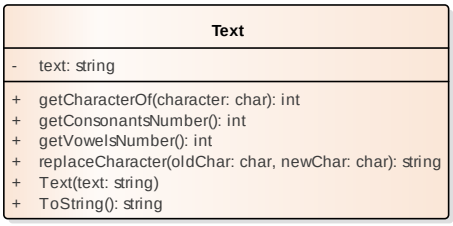
2.1.1. Osztálydiagram



2.2. A feladat ismertetése (TextApp)

Tervezzen meg, majd implementáljon egy olyan osztályt, amelynek van egy paraméteres konstruktora. A konstruktoron keresztül adjunk át egy rövidebb szöveget. A konstruktor feladata legyen az osztály adattagjának inicializálása. Az osztály példánya legyen képes a következő adatokat szolgáltatni: magánhangzók, mássalhangzók és szóközök száma. Az objektum legyen képes lecserélni a szóközöket #-re. Az egyes műveletek végrehajtásához külön függvényeket használjon, amelyeknek visszatérési típusa az adott feladattól függ. Az osztály függvényeinek a meghívását objektumon keresztül végezze. Az eredményt egy konzolos kiíratással szemléltesse.

2.2.1. Osztálydiagram

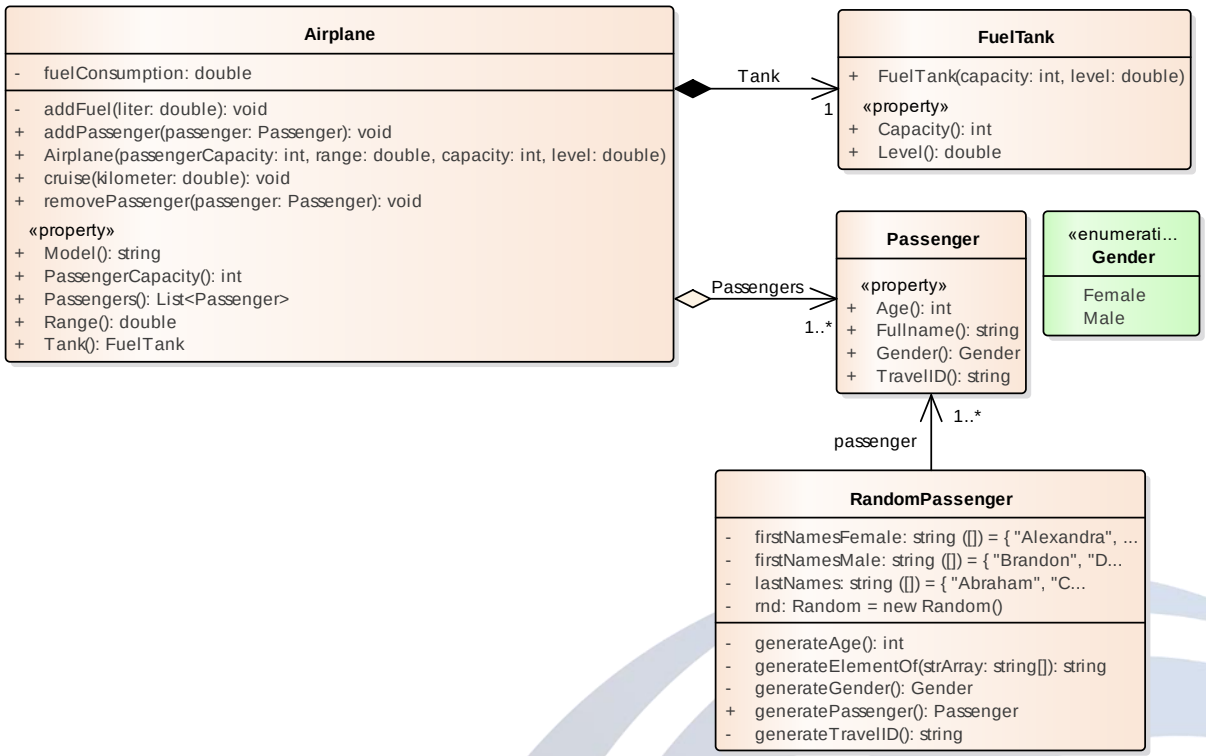


2.3. A feladat ismertetése (AirplaneApp)

Tervezzen meg, majd implementáljon repülőgépeket szimuláló kis alkalmazást. A repülőgépekre utasok szállhatnak fel, valamint le is szállhatnak azokról. A felszálláskor ügyeljünk arra, hogy egy repülőgépnek véges az utasokat befogadó képessége, valamint több utas ne szállhasson le, mint amennyi korábban felszált. Az utasok beszállását megelőzően a pilótával ismertetik az úticélt. A távolság függvényében és a fogyasztás tudatában szükséges lehet az üzemanyagtartály(ok) feltöltése, valamint arra is ügyelni kell, hogy az úticél ne legyen távolabb, mint a gép hatótávolsága. A szimuláció során egy repülőgép üzemanyagtartályának befogadó képessége, az adott repülőgép típustól függően 1000…300000 liter közé eshet. Mivel az üzemanyagtartálynak is véges a kapacitása, figyeljünk arra, hogy ne lehessen túltölteni, amikor megtelt a tartály hagyjuk abba a feltöltést. Természetesen üres üzemanyagszinttel se engedjünk

egy gépet létrehozni, a minimum elfogadott szint 100 liter. Az utasokról nyilvántartást végzünk, ahol tároljuk az utas azonosítószámát, teljes nevét, életkorát és nemét. Minden adatot véletlenszerűen generáljunk, ügyeljünk a nemnek megfelelő névválasztásra. A példányosítás során adjuk meg a repülőgép hatótávolságát kilométerben, a teljes üzemanyag kapacitását literben fejezzük ki. Az aktuális üzemanyagszint meghatározásához véletlenszámot generáljuk a minimum elfogadott üzemanyagszint és az üzemanyagtartály kapacitása között. Következő lépésben 300 és a repülőgép hatótávolsága közötti véletlenszám segítségével határozzuk meg az úticél távolságát, majd vizsgáljuk meg, hogy szükséges-e az üzemanyag-utántöltés. Ha kevés az üzemanyag, akkor vegyünk fel annyit, amennyire még szükség van az út megtételéhez. Ezt követően vegyük fel az utasokat, ahol az utasok száma 10 és a repülőgép utasokat befogadó képessége között generált véletlenszám. Utasok nélkül a járat ne induljon el. Az indulás előtt jelenítsük meg a repülőgép típusának nevét, az aktuális üzemanyagszintet és az utasok teljes nevét, majd leszállást követően ürítsük ki a gépet és jelenítsük meg a megmaradt üzemanyagmennyiséget.

2.3.1. Osztálydiagram



2.3.2. Objektumdiagram