PROGRAMOZÁS 1

5. labor

interfész (interface)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface
absztrakt módosító (abstract)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/polymorphism
Öröklődés (Inheritance)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/inheritance
Polimorfizmus (Polymorphism)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/polymorphism
virtuális módosító (virtual)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual
override módosító (override)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override
Felsorolás típus (enum)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/enum
Generikus lista (List<T> Class)
https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8
Konstruktorok (Constructors)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors
Statikus osztályok és tagok (Static Classes and Static Class Members)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members
readonly módosító (readonly)
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly
Tulajdonság (Properties)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties
String-interpoláció (string interpolation)
https://docs.microsoft.com/en-us/dotnet/api/system.io.directory?view=netframework-4.8
Hogyan írjunk szövegfájlba (How to: Write to a Text File)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-write-to-a-text-file
Hogyan olvassunk szövegfájlból (How to: Read From a Text File)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-read-from-a-text-file

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/EsO9vP2aarRFrl9Xi9Uaz6sBUihEU9kaNyO5rYXt9eDjJg?e=49vkVE

A példaprogramok letölthetők:
https://dufoffice365-my.sharepoint.com/:f:/g/personal/katonaj_uniduna_hu/EyF-Xm_Qx0hDgJKoDX2QchcBvSM3u4BYOZIr8RZNRGD6oA?e=jKi7Jh

TARTALOMJEGYZÉK

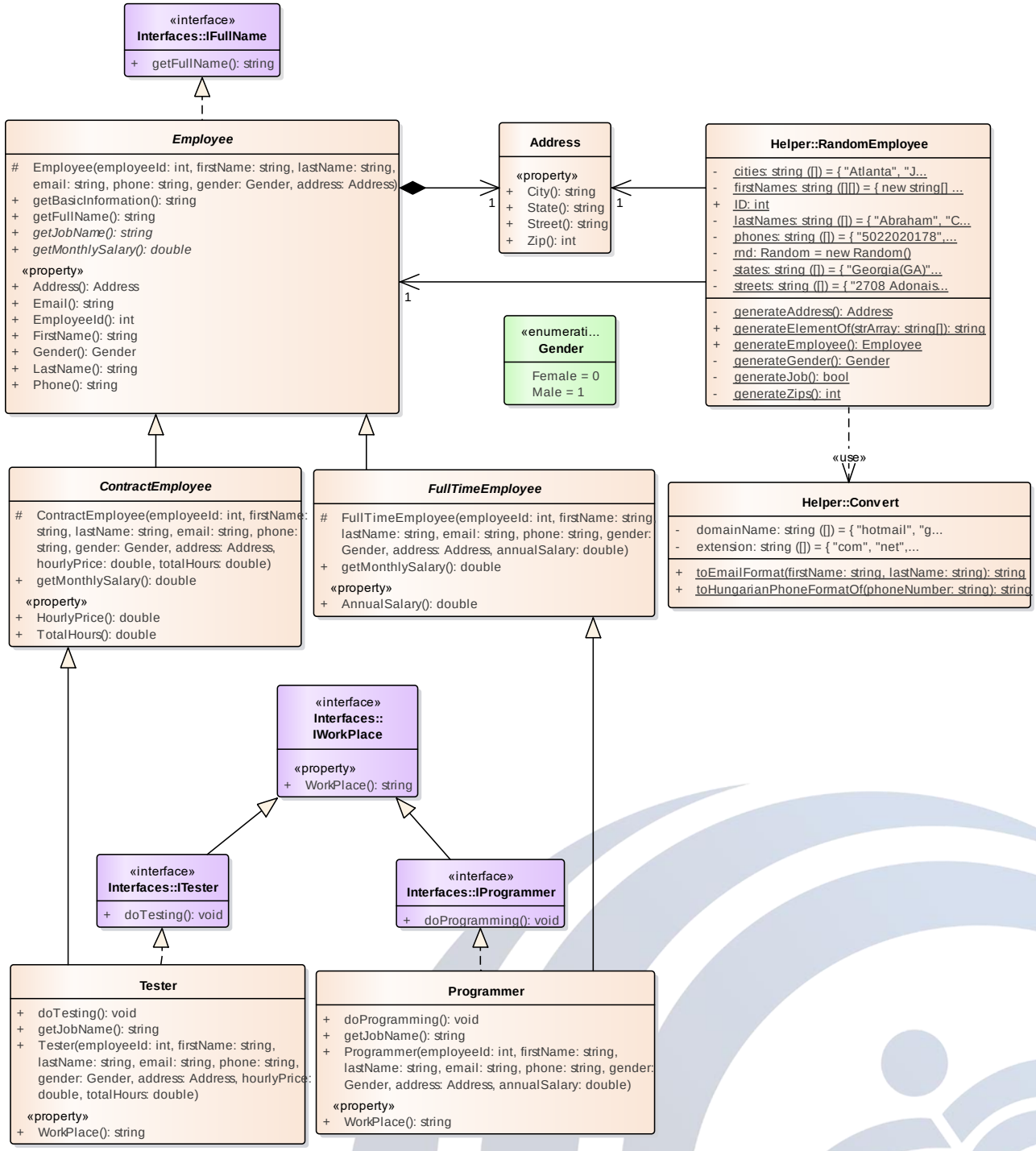
1. EmployeeApp: „Dolgozó”

1.1. A feladat ismertetése

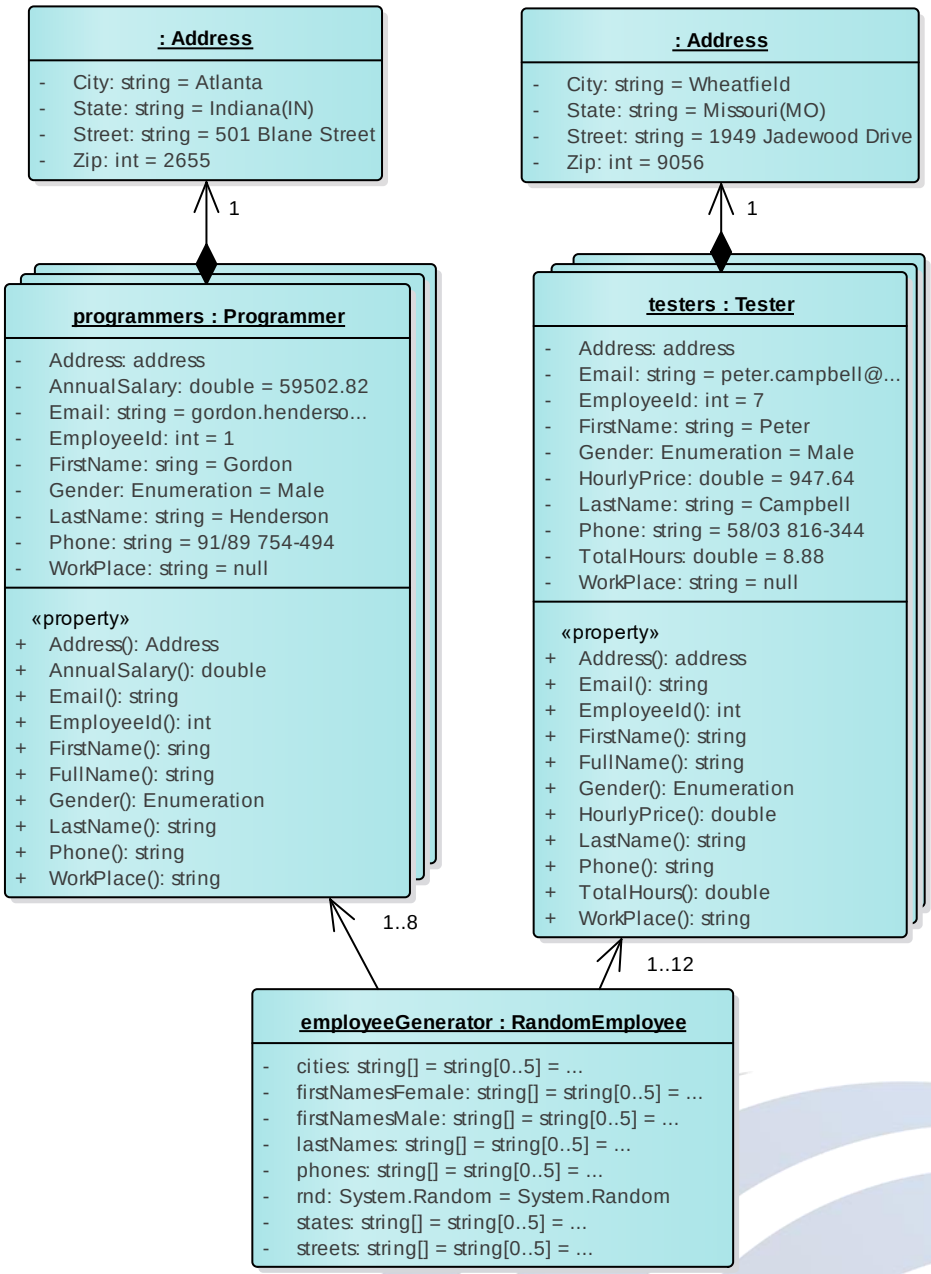
Tervezzen és készítsen el egy olyan alkalmazást, amely egy informatikai cég dolgozóinak nyilvántartó adatainak feldolgozását segítheti. A dolgozókról tárolni kell az azonosítót, vezeték- és keresztnevet, email címét, telefonszámot, nemet és lakcímet. Az alkalmazásnak képesnek kell lennie minden dolgozóról alapvető, mindenkire egységesen érvényes információkat szolgáltatni, valamint munkakör specifikusan a végzett munkát megnevezni és a hozzá kapcsolódó havi fizetést kiszámolni (ebben az esetben tekintsünk el a járulékoktól). A cégnél léteznek teljes és részmunkaidős beosztások, így a munkabér kalkulációk is eltérően történnek. Amit a követelmény feltárás során kiderítettünk, hogy a cég a programozókat teljes munkaidőben, amíg a tesztelőket részmunkaidőben foglalkoztatja. A megrendelő további kérése, hogy külön listákba kigyűjtésre kerüljenek a programozók és tesztelők. A dolgozók adatait egy szöveges fájlba is menteni szükséges. Az e-mail címek: keresztnév.vezetéknév@{hotmail|gmail|yahoo|freemail|company}.{com|net|eu|hu|uk}, amíg a telefonszámok: ##/## ###-### formában generálódjanak. A dolgozók adatai véletlenszerűen töltődjenek fel, ehhez egy külön osztályt kell implementálni. A példányosítás során egy dolgozóról el kell dönteni, hogy programozó vagy tesztelő.

1.2. A megvalósítandó alkalmazás egy lehetséges UML tervezete

1.2.1. Osztálydiagram



1.2.2. Objektumdiagram



1.3. A feladat megoldásának részletezése

A tervek alapján elkezdtük a megvalósítást. Összesen nyolc osztályra, két interfészre és egy felsorolás típusra volt szükség; *Address, ContractEmployee, Employee, FullTimeEmployee, Programmer, Tester, RandomEmployee, Convert* osztályok, *IProgrammer, ITester* interfészek és *Gender* felsorolás típus.

Az *Employee* egy absztrakt ősosztály, amely az egyes dolgozók általánosításaként született és az *IFullName* interfészből származik. A tulajdonságok több helyen is megvalósításra kerültek, így biztosítva az ellenőrzött írást és olvasást. A nem egy felsorolás típus. A lakcím esetében dekompozíció került meghatározásra, egy új osztályt (*Address*) megvalósítva. Mivel alapvetően a feladatban meghatározott

dolgozók esetében nem lehet meghatározni a pontos munkavégzést, valamint a havi keresetet, ezért azok absztrakt metódusok. A követelményeknek eleget téve a *getBasicInformation* metódus segítségével lehetőség adódik arra, hogy alapvető információk jelenjenek meg a dolgozókról.

A *FullTimeEmployee* és *ContractEmployee* absztrakt osztályok az *Employee* osztály kiterjesztései. A *FullTimeEmployee* osztály esetében a paraméteres konstruktor az éves fizetést az osztályban szereplő tulajdonságnak adja értékül, amíg a *ContractEmployee*-nál a paraméterül kapott órabért, valamint a ledolgozott órák számát az osztályban szereplő tulajdonságok kapják meg. Ezen a szinten a teljes munkaidőben alkalmazottak esetében már ismert az éves fizetés mennyisége, így meghatározható a havi fizetés is, valamint a részmunkaidősöknél ismert az óradíj és a ledolgozott órák mennyisége, így függvény felüldefiniálást alkalmazva az ősben meghatározott absztrakt metódus (*getMonthlySalary*) valósul meg.

Az *IFullName* interfész, előírja az őt megvalósító osztály számára, hogy implementáljon egy *getFullName* metódust.

Az *IProgrammer* interfész, amely az *IWorkPlace* interfészből származik és előírja az őt megvalósító osztály számára, hogy implementáljon egy *doJob* metódust.

A *Programmer* osztály a *FullTimeEmployee* osztály és az *IProgrammer* interfész leszármazottja. Az aktuálisan legalsó szinten már pontosan tudható, hogy egy programozó milyen munkát is végez, így az absztrakt metódus felüldefiniálásával megadható a végzett munka megnevezése. Az interfészben előírt metódus megvalósításával az egyes programozók utasíthatók a munkájuk megkezdésére.

Az *ITester* interfész, amely az *IWorkPlace* interfészből származik és előírja az őt megvalósító osztály számára, hogy implementáljon egy *doJob* metódust.

A *Tester* osztály a *FullTimeEmployee* osztály és az *ITester* interfész leszármazottja. Az aktuálisan legalsó szinten már pontosan tudható, hogy egy tesztelő milyen munkát is végez, így az absztrakt metódus felüldefiniálásával megadható a végzett munka megnevezése. Az interfészben előírt metódus megvalósításával az egyes tesztelők utasíthatók a munkájuk megkezdésére. Mindkét leszármazott (*Programmer*, *Tester*) osztály konstruktora az ősosztályt látja el adatokkal.

A *Gender* felsorolás típus lényegében a két ismert nemet (*Female, Male*) tárolja.

A fejlesztés során a projektben egy *Helper* nevű mappába kerültek a program helyes működéséhez szükséges osztályok (*RandomEmployee, Convert*). A *RandomEmployee* osztály leírja, hogy milyen módon generálódnak véletlenszerűen az adatok. Az osztály statikus *generateElementOf* metódusa az osztályon belül létrehozott tömbök értékeit véletlenszerűen választja ki. Az irányítószám 1 és 9999 közötti véletlenszám. A dolgozó nemének eldöntéséhez 0 és 1 között generálódik egy szám, amely, ha 0, akkor az illető nőnemű, ellenkező esetben férfi. A dolgozó munkakörének meghatározása a nem meghatározáshoz hasonlóan

történik. A lakcím generálásánál a paraméterben kapott *Address* típusú névtelen objektumpéldányt felhasználva a tulajdonságok a *generateElementOf* lokális függvényhívásokkal töltődnek fel. A dolgozó generálása (*generateEmployee*) esetében a nevet a nem határozza meg. A végzett munkatevékenységtől függően polimorfizmust alkalmazva kerül meghatározásra, hogy egy alkalmazott programozó vagy tesztelő. A konstuktor paramétereinek átadásakor a dolgozó azonosítója tekintetében egy statikus adattagot alkalmazva, az email címet és a telefonszámot a *Convert* statikus osztály *toEmailFormat,* illetve *toHungarianPhoneFormat* metódusa megfelelő formátumra alakítja. A programozók és tesztelők esetében is a havi fizetés kiszámításához szükséges adatok valamilyen értékhatár között véletlen generálódnak.

A *Program* osztály *Main* függvényében találhatók a példányok. Ezeket felhasználva meghívásra kerültek a szükséges tagfüggvények. Összességében 20 dolgozó generálódott, majd eldöntésre került, hogy ki programozó vagy tesztelő. Ettől függően töltődött fel a két lista. A generált dolgozók minden adata egy szöveges fájlba került, végül felhasználva a listákat külön csoportba sorolva jelentek meg a programozók és tesztelők neveik. A *listToConsole* függvény esetében generikus típust használunk, ahol megszorításként megfogalmazzuk, hogy a típusnak meg kell valósítania az *IFullName* interfészt.

1.4. A feladat megoldása

1.4.1. IFullName.cs

```
namespace EmployeeApp.Interfaces
{
    public interface IFullName
    {
        string getFullName();
    }
}
```

1.4.2. IWorkPlace.cs

```
namespace EmployeeApp.Interfaces
{
    public interface IWorkPlace
    {
        string WorkPlace { get; set; }
    }
}
```

1.4.3. IProgrammer.cs

```
namespace EmployeeApp.Interfaces
{
    public interface IProgrammer : IWorkPlace
    {
        void doProgramming();
    }
}
```

1.4.4. ITester.cs

```
namespace EmployeeApp.Interfaces
{
    public interface ITester : IWorkPlace
    {
        void doTesting();
    }
}
```

1.4.5. Address.cs

```
namespace EmployeeApp
{
    public class Address
    {
        public string City { get; set; }
        public string State { get; set; }
        public int Zip { get; set; }
        public string Street { get; set; }
    }
}
```

1.4.6. ContractEmployee.cs

```
namespace EmployeeApp
{
    public abstract class ContractEmployee : Employee
    {
        public double HourlyPrice { get; set; }
        public double TotalHours { get; set; }

        protected ContractEmployee(int employeeID, string firstName, string lastName,
string email,
            string phone, Gender gender, Address address, double hourlyPrice, double
totalHours)
            : base(employeeID, firstName, lastName, email, phone, gender, address)
        {
            HourlyPrice = hourlyPrice;
            TotalHours = totalHours;
        }
        public override double getMonthlySalary()
        {
            return TotalHours * HourlyPrice;
        }
    }
}
```

1.4.7. Employee.cs

```
using EmployeeApp.Interfaces;

namespace EmployeeApp
{
    public enum Gender { Female = 0, Male = 1 } // Don't change the indexes!

    public abstract class Employee : IFullName
    {
        public int EmployeeId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
```

```
        public Gender Gender { get; set; }
        public Address Address { get; set; }

        protected Employee(int employeeId, string firstName, string lastName,
            string email, string phone, Gender gender, Address address)
        {
            EmployeeId = employeeId;
            FirstName = firstName;
            LastName = lastName;
            Email = email;
            Phone = phone;
            Gender = gender;
            Address = address;
        }
        public string getBasicInformation()
        {
            return $"Employee ID : {EmployeeId} Name: {getFullName()} Email: {Email}
Address: {Address.City} {Address.State} {Address.Street} {Address.Zip}";
        }
        public string getFullName()
        {
            return FirstName + " " + LastName;
        }
        public abstract string getJobName();
        public abstract double getMonthlySalary();
    }
}
```

1.4.8. FullTimeEmployee.cs

```
namespace EmployeeApp
{
    public abstract class FullTimeEmployee : Employee
    {
        public double AnnualSalary { get; set; }

        protected FullTimeEmployee(int employeeId, string firstName, string lastName,
string email,
            string phone, Gender gender, Address address, double annualSalary)
            : base(employeeId, firstName, lastName, email, phone, gender, address)
        {
            AnnualSalary = annualSalary;
        }

        public override double getMonthlySalary()
        {
            return AnnualSalary / 12.0;
        }
    }
}
```

1.4.9. Programmer.cs

```
using System;

using EmployeeApp.Interfaces;

namespace EmployeeApp
{
    public class Programmer : FullTimeEmployee, IProgrammer
    {
        public string WorkPlace { get; set; }
```

```
        public Programmer(int employeeId, string firstName, string lastName,
            string email, string phone, Gender gender, Address address, double
annualSalary)
            : base(employeeId, firstName, lastName, email, phone, gender, address,
annualSalary)
        { }
        public override string getJobName()
        {
            return "Programmer";
        }
        public void doProgramming()
        {
            Console.WriteLine("Coding CSharp...");
        }
    }
}
```

1.4.10. Tester.cs

```
using System;

using EmployeeApp.Interfaces;

namespace EmployeeApp
{
    public class Tester : ContractEmployee, ITester
    {
        public string WorkPlace { get; set; }

        public Tester(int employeeId, string firstName, string lastName,
            string email, string phone, Gender gender, Address address,
            double hourlyPrice, double totalHours)
            : base(employeeId, firstName, lastName, email, phone, gender, address,
hourlyPrice, totalHours)
        { }
        public override string getJobName()
        {
            return "Tester";
        }
        public void doTesting()
        {
            Console.WriteLine("Testing...");
        }
    }
}
```

1.4.11. Convert.cs

```
namespace EmployeeApp.Helper
{
    class Convert
    {
        private static string[] domainName = { "hotmail", "gmail", "yahoo",
"freemail", "company" },
            extension = { "com", "net", "eu", "hu", "uk" };

        public static string toEmailFormat(string firstName, string lastName)
        {
            return
$"{firstName.ToLower()}.{lastName.ToLower()}@{RandomEmployee.generateElementOf(domainN
ame)}.{RandomEmployee.generateElementOf(extension)}";
        }
        public static string toHungarianPhoneFormatOf(string phoneNumber)
```

```
        {
            return $"{phoneNumber.Substring(0, 2)}/{phoneNumber.Substring(2, 2)}
{phoneNumber.Substring(4, 3)}-{phoneNumber.Substring(7, 4)}";
        }
    }
}
```

1.4.12. RandomEmployee.cs

```
using System;

namespace EmployeeApp.Helper
{
    public class RandomEmployee
    {
        public static int Id;

        private static Random rnd = new Random();
        private static string[][] firstNames = { new string[] { "Alexandra", "Alison",
"Maria", "Sophie", "Wanda" },
            new string[] { "Brandon", "David", "Gordon", "Jonathan", "Peter" } };
        private static string[] lastNames = { "Abraham", "Campbell", "Ellison",
"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)" },
            phones = { "50220201781", "86220893102", "61655010413", "580381634444",
"91897544945" };

        public static string generateElementOf(string[] strArray)
        {
            return strArray[rnd.Next(strArray.Length)];
        }
        private static int generateZips()
        {
            return rnd.Next(1, 10000 + 1);
        }
        private static Gender generateGender()
        {
            return rnd.Next(2) == 0 ? Gender.Female : Gender.Male;
        }
        private static bool generateJob()
        {
            return rnd.Next(2) == 0;
        }
        private static Address generateAddress()
        {
            return new Address() { City = generateElementOf(cities), State =
generateElementOf(states), Street = generateElementOf(streets), Zip = generateZips()
};
        }
        public static Employee generateEmployee()
        {
            Gender gender = generateGender();

            string firstName = generateElementOf(firstNames[(int)gender]),
                lastName = generateElementOf(lastNames);
```

```
            Employee employee;
            if (generateJob())
                employee = new Programmer(Id, firstName, lastName,
Convert.toEmailFormat(firstName, lastName),
                    Convert.toHungarianPhoneFormatOf(generateElementOf(phones)),
gender, generateAddress(),
                    rnd.NextDouble() * 100000.0);
            else
                employee = new Tester(Id, firstName, lastName,
Convert.toEmailFormat(firstName, lastName),
                    Convert.toHungarianPhoneFormatOf(generateElementOf(phones)),
gender, generateAddress(),
                    rnd.NextDouble() * 1000.0, rnd.NextDouble() * 13.0);
            ++Id;

            return employee;
        }
    }
}
```

1.4.13. Program.cs

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

using EmployeeApp.Helper;
using EmployeeApp.Interfaces;

namespace EmployeeApp
{
    class Program
    {
        static void listToConsole<T>(List<T> employees) where T : IFullName
        {
            foreach (T employee in employees)
                Console.WriteLine(employee.getFullName());
        }

        static void Main()
        {
            RandomEmployee.Id = 1;
            List<Programmer> programmers = new List<Programmer>();
            List<Tester> testers = new List<Tester>();

            using (StreamWriter file = new StreamWriter("employees.txt", true))
            {
                const string formatstr = "{0,3} {1,20} {2,6} {3,30} {4,15} {5,15}
{6,20} {7,20} {8,30} {9,10} ";
                file.WriteLine(formatstr + "{10,10}", "EID", "Name", "Gender",
"Email", "Phone", "Job",
                    "City", "State", "Street", "Zip", "Salary");

                for (int i = 0; i < 20; ++i)
                {
                    Employee employee = RandomEmployee.generateEmployee();
                    if (employee is Programmer)
                        programmers.Add(employee as Programmer);
                    else
                    {
                        Tester tester = employee as Tester;
                        testers.Add(tester);
```

```
                        //testers.Add((Tester)employee);
                    }
                    file.WriteLine(formatstr + "{10,10:#.##}", employee.EmployeeId,
employee.getFullName(),
                        employee.Gender, employee.Email, employee.Phone,
employee.getJobName(),
                        employee.Address.City, employee.Address.State,
employee.Address.Street,
                        employee.Address.Zip, employee.getMonthlySalary());
                }
            }
            Console.WriteLine("PROGRAMMERS");
            listToConsole(programmers);

            Console.WriteLine();
            Console.WriteLine("TESTERS");
            listToConsole(testers);

            Console.ReadKey();
        }
    }
}
```

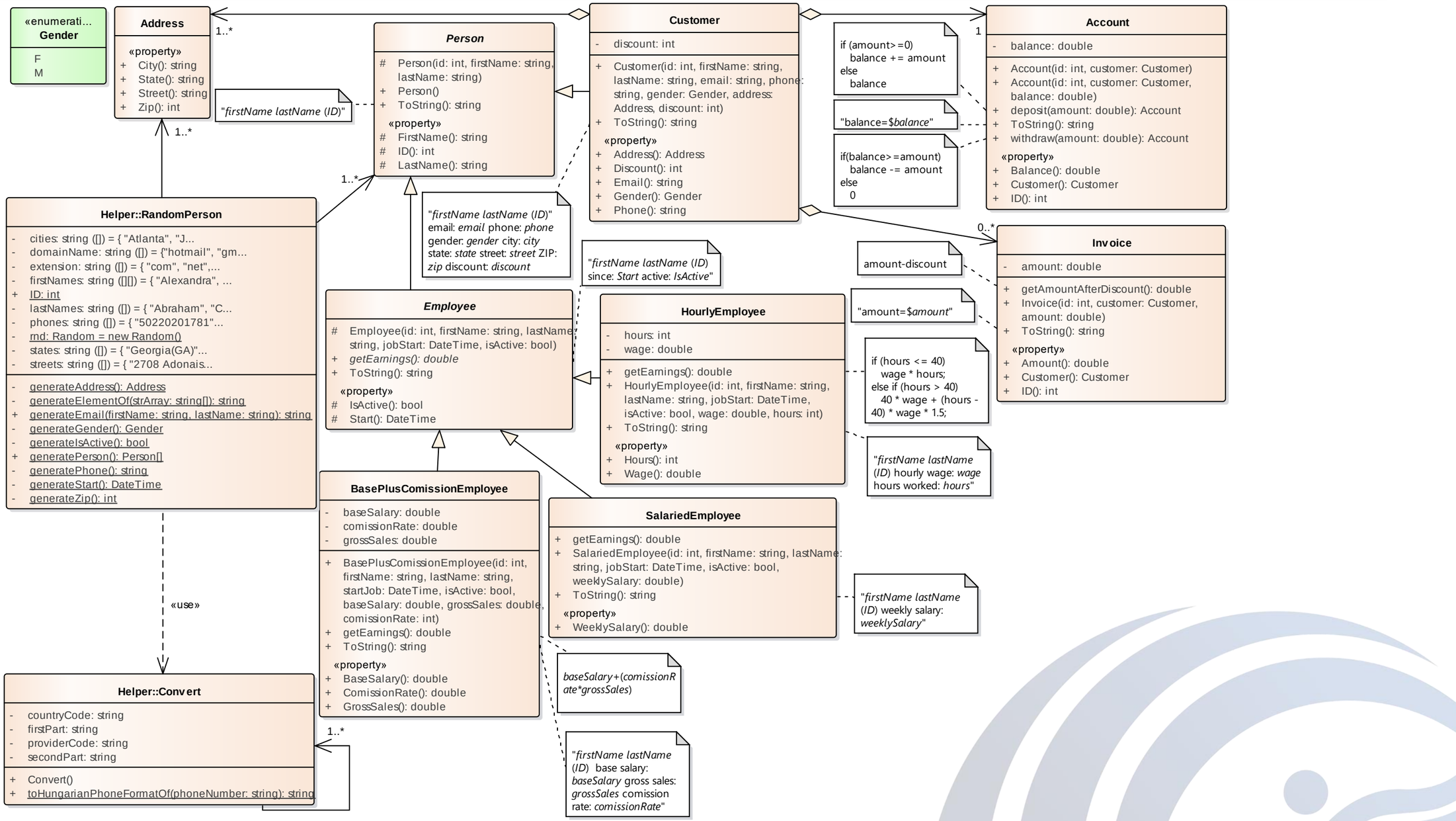
2. Gyakorló feladatok

2.1. A feladat ismertetése (EmployeeApp)

Bővítsük tovább osztályokkal és funkciókkal az órai alkalmazást. Hozzunk létre további munkaidőrendeket, mint például alkalmi munkavégzés. Bővítsük tovább a munkaköröket, mint például rendszermérnök. A korábbi órákhoz hasonlóan implementálhatunk és alkalmazhatunk saját konverziós osztályokat is. A létrehozott állapotokat mentsük ki egy szöveges fájlba.

2.2. A feladat ismertetése (CustomerApp)

Az alább látható UML osztálydiagram alapján készítsen egy objektumdiagramot, majd valósítsa meg az alkalmazást.


«enumerati...
Gender
F
M
Address
«property»
+ City(): string
+ State(): string
+ Street(): string
+ Zip(): int
1..*
"firstName lastName (ID)"
Person
Person(id: int, firstName: string, lastName: string)
+ Person()
+ ToString(): string
«property»
FirstName(): string
ID(): int
LastName(): string
Customer
- discount: int
+ Customer(id: int, firstName: string, lastName: string, email: string, phone: string, gender: Gender, address: Address, discount: int)
+ ToString(): string
«property»
+ Address(): Address
+ Discount(): int
+ Email(): string
+ Gender(): Gender
+ Phone(): string
if (amount>=0)
balance += amount
else
balance
"balance=$balance"
if(balance>=amount)
balance -= amount
else
0
Account
- balance: double
+ Account(id: int, customer: Customer)
+ Account(id: int, customer: Customer, balance: double)
+ deposit(amount: double): Account
+ ToString(): string
+ withdraw(amount: double): Account
«property»
+ Balance(): double
+ Customer(): Customer
+ ID(): int
Helper::RandomPerson
- cities: string ([]) = { "Atlanta", "J...
- domainName: string ([]) = {"hotmail", "gm...
- extension: string ([]) = { "com", "net",...
- firstNames: string ([][]) = { "Alexandra", ...
+ ID: int
- lastNames: string ([]) = { "Abraham", "C...
- phones: string ([]) = { "50220201781"...
- rnd: Random = new Random()
- states: string ([]) = { "Georgia(GA)"...
- streets: string ([]) = { "2708 Adonais...
- generateAddress(): Address
- generateElementOf(strArray: string[]): string
+ generateEmail(firstName: string, lastName: string): string
- generateGender(): Gender
- generateIsActive(): bool
+ generatePerson(): Person[]
- generatePhone(): string
- generateStart(): DateTime
- generateZip(): int
1..*
"firstName lastName (ID)" email: email phone: phone gender: gender city: city state: state street: street ZIP: zip discount: discount
"firstName lastName (ID)" since: Start active: IsActive"
Employee
Employee(id: int, firstName: string, lastName: string, jobStart: DateTime, isActive: bool)
+ getEarnings(): double
+ ToString(): string
«property»
IsActive(): bool
Start(): DateTime
HourlyEmployee
- hours: int
- wage: double
+ getEarnings(): double
+ HourlyEmployee(id: int, firstName: string, lastName: string, jobStart: DateTime, isActive: bool, wage: double, hours: int)
+ ToString(): string
«property»
+ Hours(): int
+ Wage(): double
if (hours <= 40)
wage * hours;
else if (hours > 40)
40 * wage + (hours - 40) * wage * 1.5;
"firstName lastName (ID) hourly wage: wage hours worked: hours"
Invoice
- amount: double
+ getAmountAfterDiscount(): double
+ Invoice(id: int, customer: Customer, amount: double)
+ ToString(): string
«property»
+ Amount(): double
+ Customer(): Customer
+ ID(): int
amount-discount
"amount=$amount"
BasePlusComissionEmployee
- baseSalary: double
- comissionRate: double
- grossSales: double
+ BasePlusComissionEmployee(id: int, firstName: string, lastName: string, startJob: DateTime, isActive: bool, baseSalary: double, grossSales: double, comissionRate: int)
+ getEarnings(): double
+ ToString(): string
«property»
+ BaseSalary(): double
+ ComissionRate(): double
+ GrossSales(): double
SalariedEmployee
+ getEarnings(): double
+ SalariedEmployee(id: int, firstName: string, lastName: string, jobStart: DateTime, isActive: bool, weeklySalary: double)
+ ToString(): string
«property»
+ WeeklySalary(): double
"firstName lastName (ID) weekly salary: weeklySalary"
baseSalary+(comissionRate*grossSales)
"firstName lastName (ID) base salary: baseSalary gross sales: grossSales comission rate: comissionRate"
«use»
Helper::Convert
- countryCode: string
- firstPart: string
- providerCode: string
- secondPart: string
+ Convert()
+ toHungarianPhoneFormatOf(phoneNumber: string): string
1..*