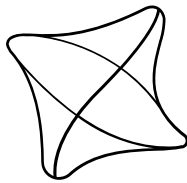


2



Práca s triedou

Ing. Štefan Toth, PhD.

Obsah cvičenia

- Vytvorenie **triedy** (class)
- Používanie **vlastností** (properties)
- **Preťažovanie** (overloading) konštruktorov a metód
- **Voliteľné argumenty** (optional arguments) konštruktorov a metód
- **Prekrytie** (overriding) metód **ToString()**, **Equals()**, **GetHashCode()**

Úloha 1

- **Vytvorte konzolovú aplikáciu, v ktorej implementujte postupne nasledujúce typy:**

Person
Class

Fields

- `_firstName : string`

Properties

- `Age { get; } : int`
- `Birthday { get; set; } : DateTime?`
- `FirstName { get; set; } : string`
- `FullName { get; } : string`
- `Gender { get; set; } : Gender`
- `LastName { get; set; } : string`

Methods

- `Equals(object obj) : bool`
- `GetHashCode() : int`
- `Person()`
- `Person(string firstName, string lastName, DateTime? birthday, [Gender gender = Gender.Unknown])`
- `ToString() : string`

Gender
Enum

- Unknown
- Male
- Female

PersonDatabase
Class

Fields

- `_people : List<Person>`

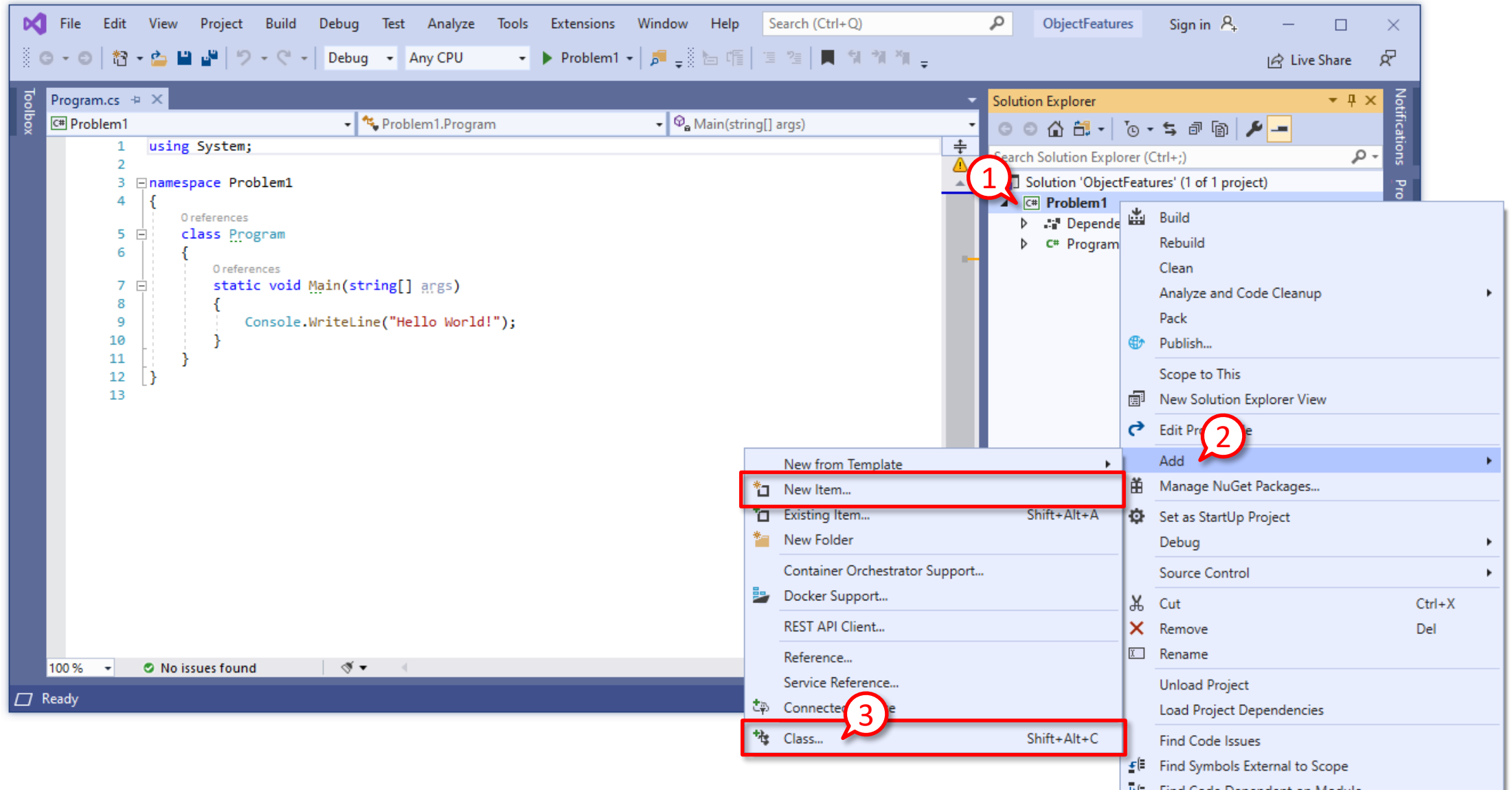
Methods

- `Add(params Person[] people) : void`
- `Add(Person person) : void`
- `Find(string text, [Gender? gender = null]) : List<Person>`
- `PrintToConsole() : void`
- `Remove(Person person) : void`

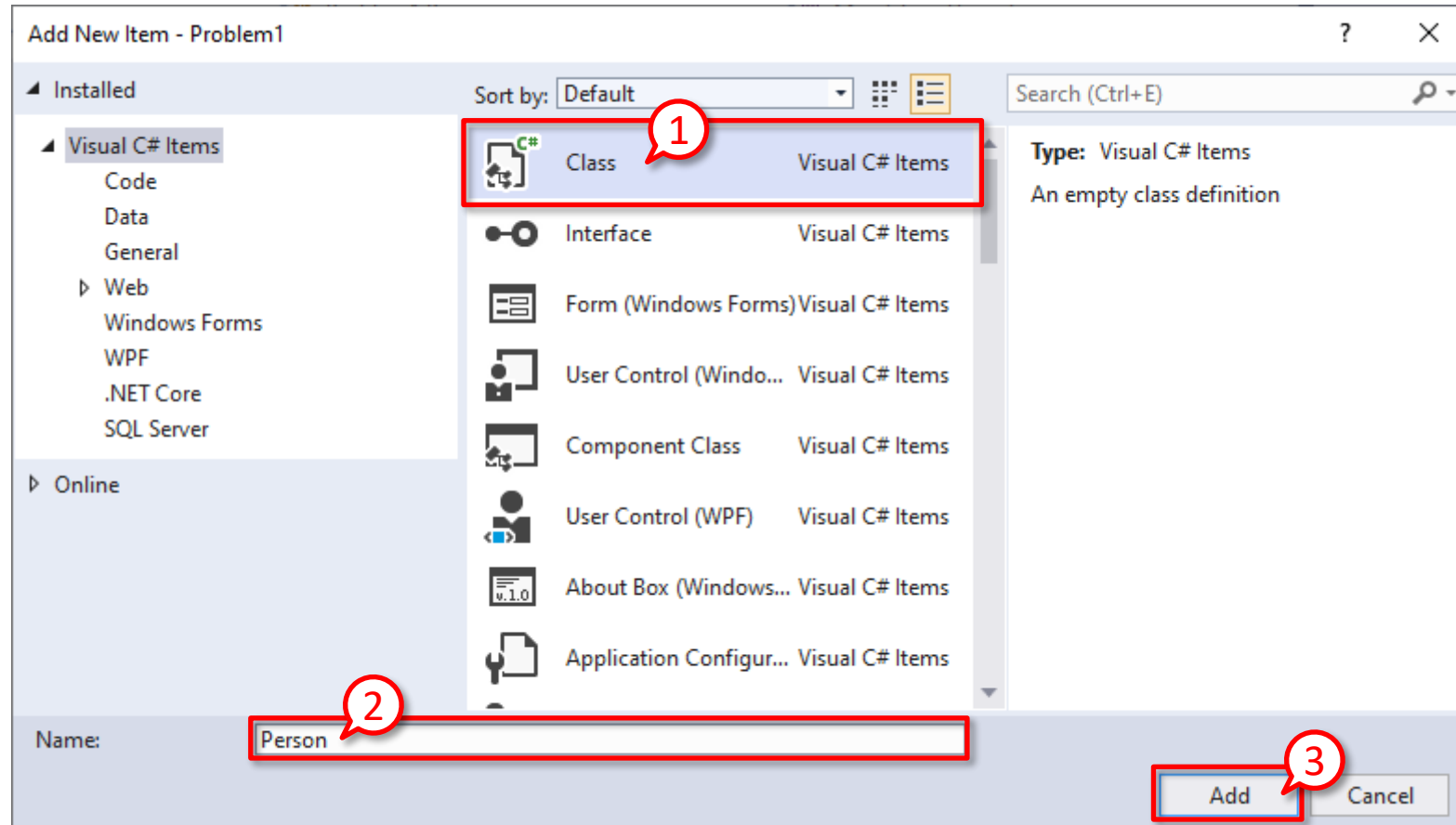
Úloha 1.1 (trieda Person)

- V projekte **pridajte triedu Person** (Osoba), ktorá bude **obsahovať** tieto **vlastnosti** (property):
 - **FirstName** (Meno)
 - **LastName** (Priezvisko)
 - **FullName** (Meno a priezvisko) – len na čítanie bez dátového člena
 - **Birthday** (Dátum narodenia)
 - **Age** (Vek) – bude sa automaticky vypočítavať z dátumu narodenia a aktuálneho času
- Vyskúšajte si rôzne spôsoby tvorby vlastností (s dát. členmi, automatické, ...):
<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties>

Pridanie novej triedy (cez New Item... alebo Class...)



Pridanie novej triedy (Class – názov - Add)



Pridanie vlastnosti

- Snippet **prop** vytvorí v triede **automatickú vlastnosť**, tabulátorom sa prepíname medzi typom a názvom vlastnosti, ktoré môžeme modifikovať

```
1  using System;
2      using System.Collections.Generic;
3      using System.Text;
4
5  namespace Problem1
6  {
7      0 references
8      class Person
9      {
10         0 references
11         public int MyProperty { get; set; }
12     }
```

Príklady definovania vlastnosti FirstName

Vlastnosť (full property):

```
// Súkromný dátový člen (field)
private string _firstName;

// Verejná vlastnosť zaobalujúca prístup
// k dátovému členu (staršia syntax):
public string FirstName
{
    get { return _firstName; }
    set { _firstName = value; }
}

// Alebo je možné jej get a set časť
// definovať aj cez lambda operátor (=>)
public string FirstName
{
    get => _firstName;
    set => _firstName = value;
}
```

Automatická vlastnosť (auto property):

```
// Verejná vlastnosť na čítanie i zapisovanie
public string FirstName { get; set; }
```

```
// Verejná vlastnosť na čítanie, avšak
// súkromná na zapisovanie (iba vo vnútri triedy)
public string FirstName { get; private set; }
```

```
// Verejná vlastnosť iba na čítanie,
// nastavovať a zapisovať do nej sa dá iba v
// konštruktore:
public string FirstName { get; }
```

```
// alebo aj priamo pri inicializovaní:
public string FirstName { get; } = "Jano";
```


Vlastnosti – rôzne zápisy (vývoj jazyka C#)

Vlastnosť FullName definovaná 3 rôznymi spôsobmi (všetky sú ekvivalentné):

```
// Súkromný dátový člen (field)
private string _fullName;

// (1. spôsob) Verejná vlastnosť
// iba na čítanie:
public string FullName
{
    get { return _fullName; }
}

// (2. spôsob) Cez lambda operátor (=>),
// return a zátvorky { } sa vynechávajú
public string FullName
{
    get => _fullName;
}

// (3. spôsob) Vynechanie get
public string FullName => _fullName;
```

```
public string FirstName { get; set; }

public string LastName { get; set; }

// Bez použitia dátového člena _fullName, výsledkom
// je reťazec zložený z mena a priezviska
public string FullName => $"{FirstName} {LastName}";
```

Rýchle akcie (Quick Actions)

- Klávesová skratka **Ctrl + .** (alebo **Alt + Enter**) nad identifikátorom vlastnosti:

The screenshot illustrates the process of converting a property to an auto-property in Visual Studio. It shows two states of the code editor:

Initial State (Left): The code defines a `public int FirstName` property with a private backing field `_firstName`. A context menu is open over the `FirstName` identifier, with **Use auto property** highlighted. A red box highlights this option. A red arrow points from this option to the next state.

Final State (Right): The code now shows the `FirstName` property as an auto-property: `public int FirstName { get; set; }`. A second context menu is open over this property, with **Convert to full property** highlighted. A red box highlights this option. A red arrow points from this option back to the initial state, indicating the reverse action.

The code snippets shown are:

```
Initial State (Left):
public int FirstName
{
    private int _firstName;

    public int FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
}

Final State (Right):
public int FirstName { get; set; }
```

Úloha 1.2 (vytvorenie inštancie Person)

- V metóde Main() **vytvorte ľubovoľný objekt typu Person**, napr.:

```
var jano1 = new Person();  
jano1.FirstName = "Ján";  
jano1.LastName = "Mrkvička";  
jano1.Gender = Gender.Male;
```

alebo **pomocou inicializátora objektu** (volá konštruktor + nastavuje vlastnosti):

```
var jano2 = new Person { FirstName = "Ján",  
                          LastName = "Mrkvička",  
                          Gender = Gender.Male };
```

- Dokumentácia pre inicializátory:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/object-and-collection-initializers>

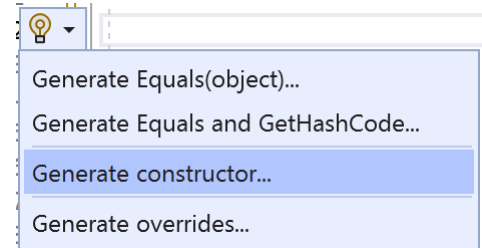
Úloha 1.3 (parametrický konštruktor)

- **Pridajte parametrický konštruktor**, ktorý bude inicializovať objekt s parametrami mena, priezviska, dátumu narodenia a pohlavia, aby ste mohli vytvoriť objekt jednoduchšie cez konštruktor:

```
var jano3 = new Person("Ján", "Mrkvička", new DateTime(1985, 12, 31), Gender.Male);
```

- Vytvorený objekt **osoby vypíšte na obrazovku** pomocou **formátovacieho reťazca** v tvare

"{FullName}, {Birthday:dd.MM.yyyy}, age: ({Age}), gender: {Gender}"



Úloha 1.4 (voliteľný parameter a nullable)

- Upravte konštruktor tak, aby:
 - parameter **pohlavie** bol voliteľný a implicitne nastavený na neznáme pohlavie,
 - parameter **dátum narodenia** mohol nadobúdať **null hodnoty**, navyše ak bude parameter mať hodnotu **null**, nastavte vlastnosť na aktuálny dátum.

```
var jano4 = new Person("Ján", "Mrkvička", null);
```

- Ďalšie informácie v dokumentácii:
 - Pomenované a voliteľné argumenty:
<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments>
 - Nullable hodnotové typy:
<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types>

Úloha 1.5 (Equals)

- Pridajte do triedy metódu **Equals** pre zistenie, či je objekt rovný s iným objektom:

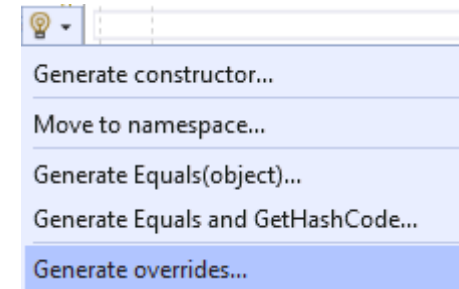
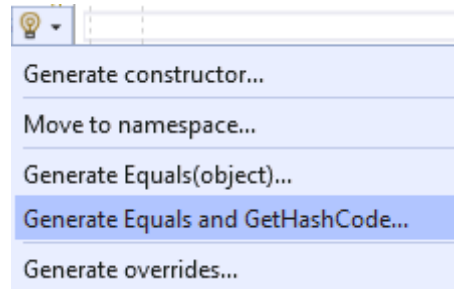
```
var jano2 = new Person
{
    FirstName = "Ján",
    LastName = "Mrkvička",
    Gender = Gender.Male,
    Birthday = DateTime.Parse("31.12.1985", new CultureInfo("sk"))
};

var jano3 = new Person("Ján", "Mrkvička", new DateTime(1985, 12, 31), Gender.Male);

bool result = jano2.Equals(jano3); // result == False (Prečo?)
```

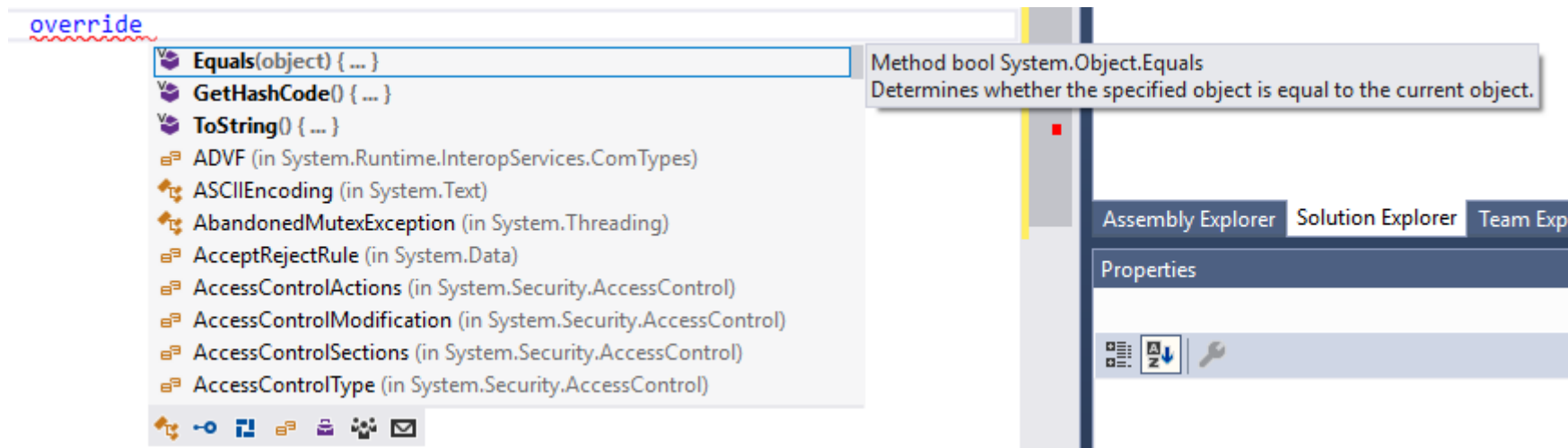
Úloha 1.6 (prekryte ToString, Equals, GetHashCode)

- **Prekryte** (override) v triede Person **nasledujúce metódy** predka **object** (System.Object):
 - **ToString()** – bude vypisovať osobu vo formáte „Meno Priezvisko (Vek)“
 - **Equals()** – bude porovnávať objekty typu Person (objekty budú rovné, ak sa zhodujú vo všetkých hodnotách vlastností). S metódou je nutné prekryť aj metódu **GetHashCode()**, ktorá vracia číslo identifikujúca objekty, najmä pre hash kolekcie
- Na vygenerovanie môžete použiť Visual Studio – na prázdnom riadku použite **Ctrl + .** (alebo **Alt + Enter**):



Prekrývanie (Visual Studio – IntelliSense)

- Ďalším spôsobom, ako prekryť metódy, je napísať kľúčové slovo **override** v triede, IntelliSense následne zobrazí všetky možné metódy, ktoré je možné prekryť z predkov
- Príklad zobrazuje 3 metódy, ktoré sú virtuálne v predkovi (System.Object) a neboli ešte prekryté:



Zoznámte sa s najpoužívanějšími kolekciami

- IList<T>
- ICollection<T>
- IEnumerable<T>
- IEnumerable
- IList
- ICollection
- ReadOnlyList<T>
- ReadOnlyCollection<T>

List<T>
Generic Class

Properties

- Capacity { get; set; } : int
- Count { get; } : int
- this[int index] { get; set; } : T

Methods

- Add(T item) : void
- AddRange(IEnumerable<T> collection) : void
- AsReadOnly() : ReadOnlyCollection<T>
- BinarySearch(int index, int count, T item, IComparer<T> comparer) : int
- BinarySearch(T item) : int
- BinarySearch(T item, IComparer<T> comparer) : int
- Clear() : void
- Contains(T item) : bool
- ConvertAll<TOutput>(Converter<T, TOutput> converter) : List<TOutput>
- CopyTo(int index, T[] array, int arrayIndex, int count) : void
- CopyTo(T[] array) : void
- CopyTo(T[] array, int arrayIndex) : void
- Exists(Predicate<T> match) : bool
- Find(Predicate<T> match) : T
- FindAll(Predicate<T> match) : List<T>
- FindIndex(int startIndex, int count, Predicate<T> match) : int
- FindIndex(int startIndex, Predicate<T> match) : int
- FindIndex(Predicate<T> match) : int
- FindLast(Predicate<T> match) : T
- FindLastIndex(int startIndex, int count, Predicate<T> match) : int
- FindLastIndex(int startIndex, Predicate<T> match) : int
- FindLastIndex(Predicate<T> match) : int

- ICollection<T>
- IEnumerable<T>
- IEnumerable
- ISerializable
- IDeserializationCallback
- ISet<T>
- ReadOnlyCollection<T>

HashSet<T>
Generic Class

Properties

- Comparer { get; } : IEqualityComparer<T>
- Count { get; } : int

Methods

- Add(T item) : bool
- Clear() : void
- Contains(T item) : bool
- CopyTo(T[] array) : void
- CopyTo(T[] array, int arrayIndex) : void
- CopyTo(T[] array, int arrayIndex, int count) : void
- CreateSetComparer() : IEqualityComparer<HashSet<T>>
- ExceptWith(IEnumerable<T> other) : void
- GetEnumerator() : Enumerator
- GetObjectData(SerializationInfo info, StreamingContext context) : void
- HashSet()
- HashSet(IEnumerable<T> collection)
- HashSet(IEnumerable<T> collection, IEqualityComparer<T> comparer)
- HashSet(IEqualityComparer<T> comparer)
- HashSet(SerializationInfo info, StreamingContext context)
- IntersectWith(IEnumerable<T> other) : void
- IsProperSubsetOf(IEnumerable<T> other) : bool
- IsProperSupersetOf(IEnumerable<T> other) : bool
- IsSubsetOf(IEnumerable<T> other) : bool
- IsSupersetOf(IEnumerable<T> other) : bool
- OnDeserialization(object sender) : void
- Overlaps(IEnumerable<T> other) : bool
- Remove(T item) : bool

- IDictionary<TKey, TValue>
- ICollection<KeyValuePair<TKey, TValue>>
- IEnumerable<KeyValuePair<TKey, TValue>>
- IEnumerable
- IDictionary
- ICollection
- ReadOnlyDictionary<TKey, TValue>
- ReadOnlyCollection<KeyValuePair<TKey, TValue>>
- ISerializable
- IDeserializationCallback

Dictionary<TKey, TValue>
Generic Class

Properties

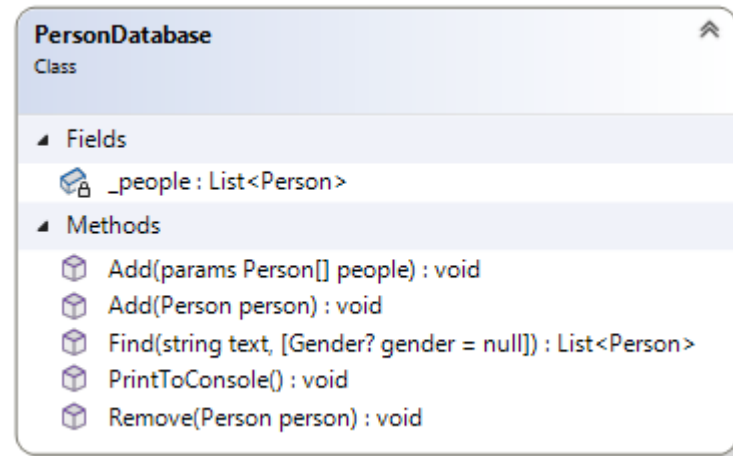
- Comparer { get; } : IEqualityComparer<TKey>
- Count { get; } : int
- Keys { get; } : KeyCollection
- this[TKey key] { get; set; } : TValue
- Values { get; } : ValueCollection

Methods

- Add(TKey key, TValue value) : void
- Clear() : void
- ContainsKey(TKey key) : bool
- ContainsValue(TValue value) : bool
- Dictionary()
- Dictionary(IDictionary<TKey, TValue> dictionary)
- Dictionary(IDictionary<TKey, TValue> dictionary, IEqualityComparer<TKey> comparer)
- Dictionary(IEqualityComparer<TKey> comparer)
- Dictionary(int capacity)
- Dictionary(int capacity, IEqualityComparer<TKey> comparer)
- Dictionary(SerializationInfo info, StreamingContext context)
- GetEnumerator() : Enumerator
- GetObjectData(SerializationInfo info, StreamingContext context) : void
- OnDeserialization(object sender) : void
- Remove(TKey key) : bool
- TryGetValue(TKey key, out TValue value) : bool

Úloha 1.7 (PersonDatabase)

- **Vytvorte** a implementujte **triedu PersonDatabase**:

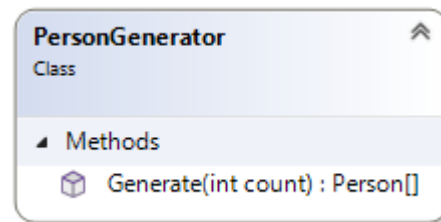


- Pre oboznámenie sa s kolekciami, môžete dátový člen `_people` definovať s použitím kolekcie **HashSet<T>** alebo **Dictionary<T>** namiesto **List<T>**

Úloha 1.8 (PersonDatabase)

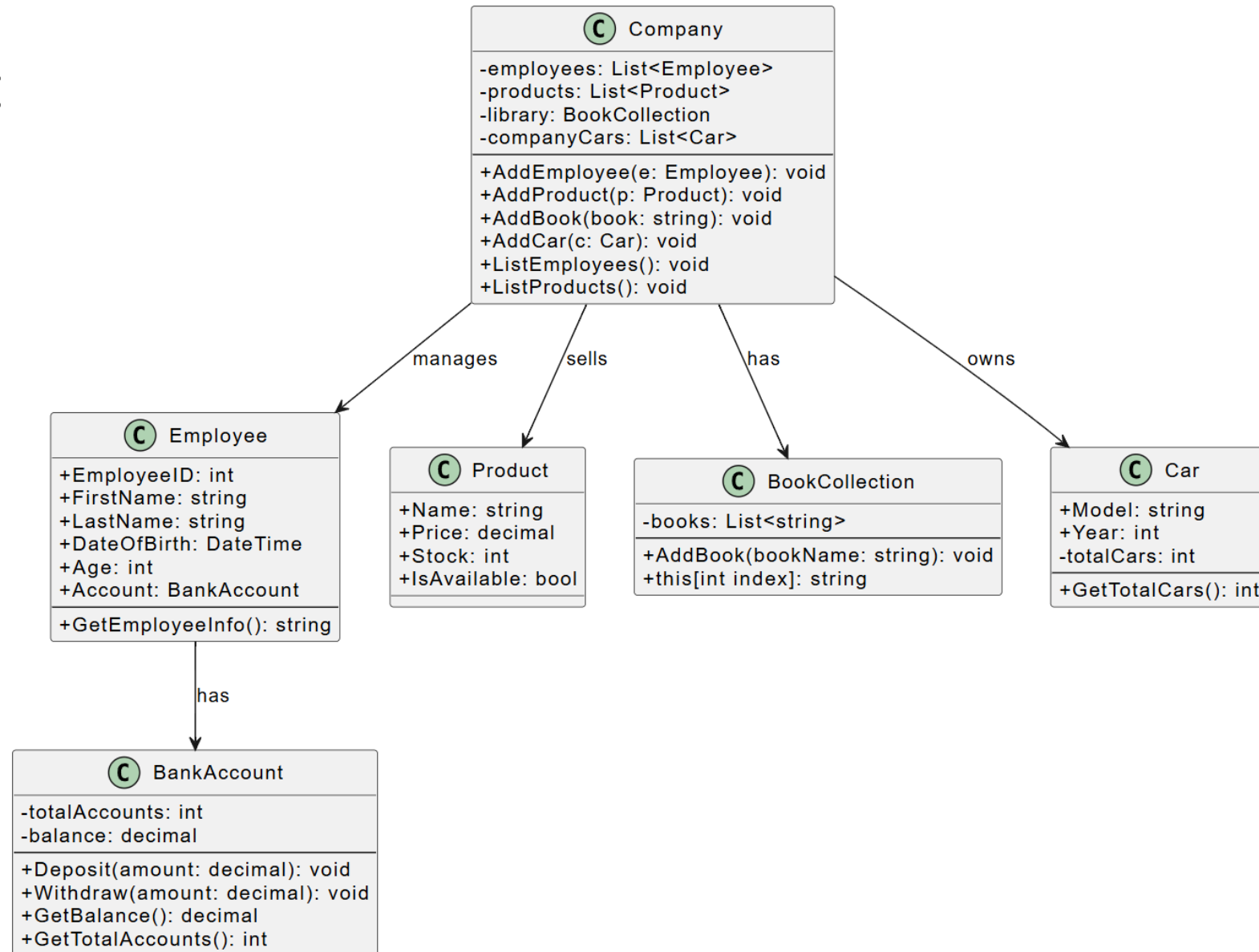
- Vytvorte triedu **PersonGenerator**, ktorá bude generovať osoby napr. z takýchto polí:

```
{ "Ján", "Alexander", "Adam", "Juraj", "Štefan" }  
{ "Nový", "Malý", "Veľký", "Chudý", "Vysoký", "Bohatý", "Krásny" }  
  
{ "Silvia", "Františka", "Michaela", "Barbora", "Eva" }  
{ "Nová", "Malá", "Veľká", "Chudá", "Vysoká", "Bohatá", "Krásna" }
```



Úloha 2

- Vytvorte si nový projekt konzolovej aplikácie a implementujte jednotlivé triedy podľa zadania



Zdroje

- .NET Documentation
<https://learn.microsoft.com/en-us/dotnet/>
- C# documentation
<https://learn.microsoft.com/en-us/dotnet/csharp/>
- Visual Studio product family documentation
<https://learn.microsoft.com/en-us/visualstudio/>

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu **Jazyk C# a .NET** na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Ing. **Štefan Toth**, PhD.
stefan.toth@uniza.sk