LAB 4

*Version: 0.1*

*Date: 2018.09.01*

*Author: Vinh Hoang*

**Part I: Defining Classes**

**1.   Define Bank Account Class**

Create a class named BankAccount.

The class should have private fields for:

* id: int
* balance: double

The class should also have public properties for:

* ID:int
* Balance:double

Create a **new class** and ensure **proper naming**

**2.   Methods**

Create a class BankAccount (you can use class from previous task)

The class should have private fields for:

* id: int
* balance: double

The class should also have properties for:

* ID:int
* Balance:double
* Deposit(Double amount):void
* Withdraw(Double amount):void

Override the method ToString().

You should be able to use the class like this:

**Solution**

Create a method Deposit(double amount)

Create a method Withdraw(double amount)

Override the method toString()

**3.   Test Client**

Create a test client that tests your BankAccount class.

Support the **following commands**:

* **Create {Id}**
* **Deposit {Id} {Amount}**
* **Withdraw {Id} {Amount}**
* **Print {Id}**
* **End**

If you try to create an account with existing Id, print **"Account already exists".**

If you try to perform an operation on **non-existing account** with existing Id, print **"Account does not exist"**.

If you try to withdraw an amount larger than the balance, print **"Insufficient balance"**.

The Print command should print **"Account ID{id}, balance {balance}"**. Round the balance to the second digit after the decimal separator.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Create 1  Create 2  Deposit 1 20  Withdraw 1 30  Withdraw 1 10  Print 1  End | Account already exists  Insufficient balance  Account ID1, balance 10.00 |
| Create 1  Deposit 2 20  Withdraw 2 30  Print 2  End | Account does not exist  Account does not exist  Account does not exist |

**Solution**

Create a Dictionary<int, BankAccount> to store existing accounts

Create the input loop

Check the **type of command** and **execute** accordingly (***optional:****you can create a separate method for each command*)

Implement the Create command

Implement the rest of the commands following the same logic

**4.   Define Person Class**

Create a **Person** class.

The class should have **private fields** for:

* Name: **string**
* Age: **int**
* Accounts: List<BankAccount>

The class should have **constructors**:

* Person(string name, int age)
* Person(string name, int age, List<BankAccount> accounts)

The class should also have **public methods** for:

* GetBalance():double

**5.   Sort Persons by Name and Age**

Create a class **Person**, which should have **private** fields for:

* FirstName: string
* LastName: string
* Age: int
* ToString(): string - override

You should be able to use the class like this:

|  |
| --- |
| StartUp.cs |
| public static void Main()  {      var lines = int.Parse(Console.ReadLine());      var persons = new List<Person>();      for (int i = 0; i < lines; i++)      {          var cmdArgs = Console.ReadLine().Split();          var person = new Person(cmdArgs[0], cmdArgs[1], int.Parse(cmdArgs[2]));          persons.Add(person);      }        persons.OrderBy(p => p.FirstName)             .ThenBy(p => p.Age)             .ToList()             .ForEach(p => Console.WriteLine(p.ToString()));  } |

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Asen Ivanov 65  Boiko Borisov 57  Ventsislav Ivanov 27  Asen Harizanoov 44  Boiko Angelov 35 | Asen Harizanoov is a 44 years old.  Asen Ivanov is a 65 years old.  Boiko Angelov is a 35 years old.  Boiko Borisov is a 57 years old.  Ventsislav Ivanov is a 27 years old. |

**Solution**

Create a **new class** and ensure **proper naming**. Define the **private** fields

Create a constructor for Person, which takes 3 parameters firstName, lastName, age.

Create properties for these fields, which are as strictly as possible

Override **ToString()** method:

**6.   Salary Increase**

**Refactor project from last task.**

Read person with their names, age and salary. Read percent bonus to every person salary. Persons younger than 30 get half bonus. Expand **Person** from previous task. New **fields** and **methods:**

* Salary: double
* IncreaseSalary(double bonus)

You should be able to use the class like this:

|  |
| --- |
| StartUp.cs |
| var lines = int.Parse(Console.ReadLine());  var persons = new List<Person>();  for (int i = 0; i < lines; i++)  {      var cmdArgs = Console.ReadLine().Split();      var person = new Person(cmdArgs[0],                              cmdArgs[1],                              int.Parse(cmdArgs[2]),                              double.Parse(cmdArgs[3]));        persons.Add(person);  }  var bonus = double.Parse(Console.ReadLine());  persons.ForEach(p => Console.WriteLine(p.ToString())); |

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Asen Ivanov 65 2200  Boiko Borisov 57 3333  Ventsislav Ivanov 27 600  Asen Harizanoov 44 666.66  Boiko Angelov 35 559.4  20 | Asen Ivanov get 2200 leva  Boiko Borisov get 3333 leva  Ventsislav Ivanov get 600 leva  Asen Harizanoov get 666.66 leva  Boiko Angelov get 559.4 leva |

**Solution**

Add new **private** field for **salary** and **refactor constructor**. Add new **method**, which will **update** salary with bonus

Refactor **toString()** method for this task.

**7.   Validation Data**

Expand Person with proper validation for every field:

* **Names must be at least 3 symbols**
* **Age must not be zero or negative**
* **Salary can't be less than 460.0**

Print proper message to end user (look at example for messages).

Use ArgumentExeption with messages from example.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Asen Ivanov -6 2200  B Borisov 57 3333  Ventsislav Ivanov 27 600  Asen H 44 666.66  Boiko Angelov 35 300  20 | Age cannot be zero or negative integer  First name cannot be less than 3 symbols  Last name cannot be less than 3 symbols  Salary cannot be less than 460 leva  Ventsislav Ivanov get 660.0 leva |

**8.   First and Reserve Team**

Create a Team class. Add to this team all person you read. All person younger than 40 go in first team, others go in reverse team. At the end print first and reserve team sizes.

The class should have **private fields** for:

* name: string
* firstTeam: List<Person>
* reserveTeam: List<Person>

The class should have **constructors**:

* Team(string name)

The class should also have **public methods** for:

* AddPlayer(Person person): void
* FirstTeam: IReadOnlyCollection
* ReserveTeam: IReadOnlyCollection

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Asen Ivanov 20 2200  Boiko Borisov 57 3333  Ventsislav Ivanov 27 600  Grigor Dimitrov 25 666.66  Boiko Angelov 35 555 | First team have 4 players  Reserve team have 1 players |