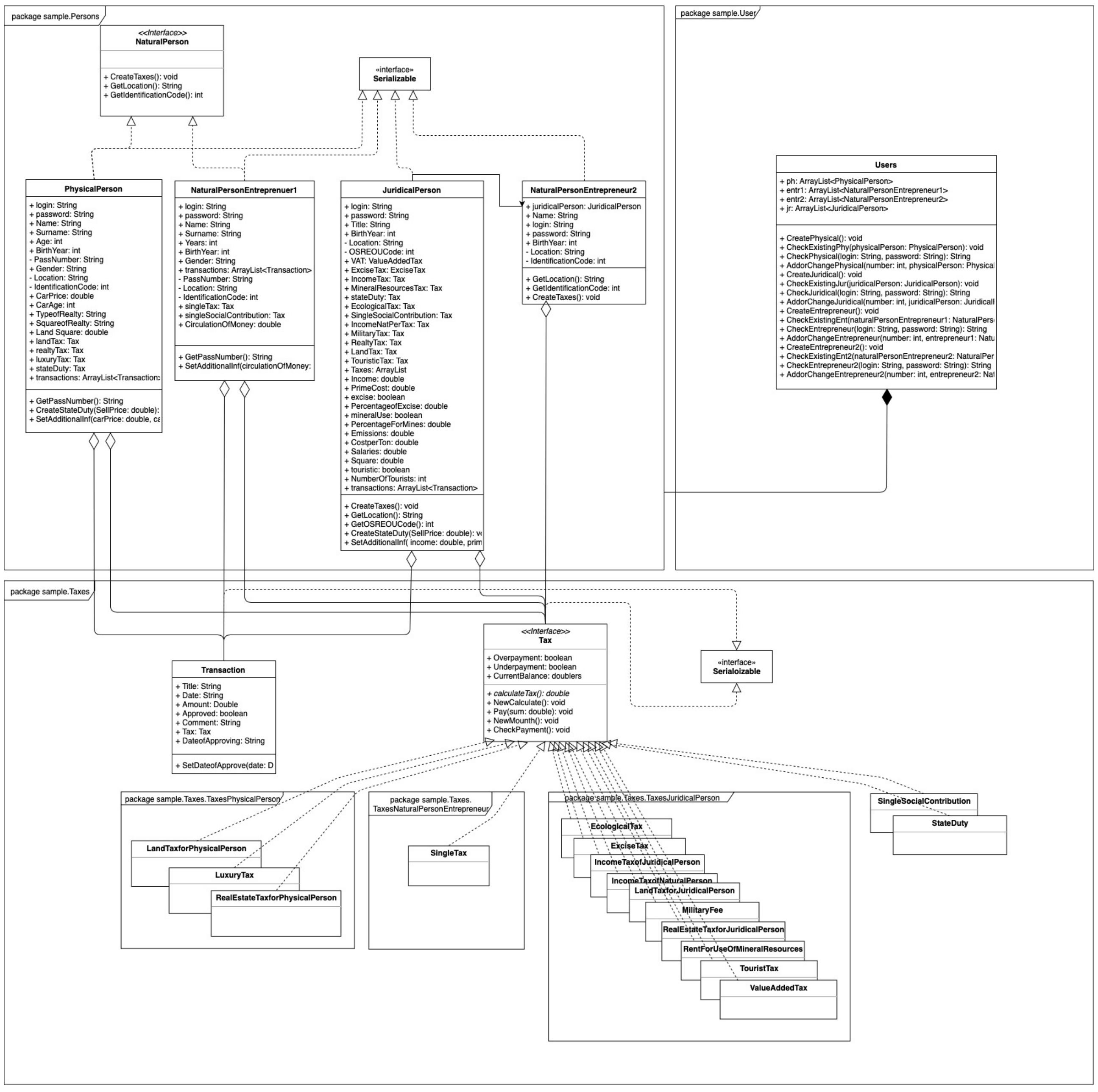
Name and Surname: Olesia Melnychyn
Project title: E-Taxpayer
Project Objective:

«E-Taxpayer» is an application made for every person. It is adapted for each of taxpayers due to his/her functions and possibilities. «E-Taxpayer» is a very useful application, which helps you to regulate almost all your taxes just sitting at your computer. There is no more long queues and doubtful sums. It provides you a possibility to see all you taxes as a list, which is constructed especially for each natural person, natural person-entrepreneur and juridical person.

Firstly, you have to register yourself in the application. And to see all the stuff, you have to add some additional information about your job. The main functions, which help you not to lost in your calculation, are: all payments are listed and you can also see if there is any over-/underpayment on every tax. Also there is a possibility to add a single, not periodic action, which cause taxpaying (like selling car or realty). Moreover, you can construct different lists of taxes by their types (for example: ones which have under-/ overpayment, ones which should be paid every month). Also there may be a list with your payments in date order and a total sum paid to the country.

«E-Taxpayer» is an application which puts in order all your spendings as a ratepayer.

Structure:



package sample.Persons
<<Interface>> NaturalPerson
+ CreateTaxes(): void
+ GetLocation(): String
+ GetIdentificationCode(): int
«interface» Serializable
PhysicalPerson
+ login: String
+ password: String
+ Name: String
+ Surname: String
+ Age: int
+ BirthYear: int
- PassNumber: String
+ Gender: String
- Location: String
- IdentificationCode: int
+ CarPrice: double
+ CarAge: int
+ TypeofRealty: String
+ SquareofRealty: String
+ Land Square: double
+ landTax: Tax
+ realtyTax: Tax
+ luxuryTax: Tax
+ stateDuty: Tax
+ transactions: ArrayList<Transaction>
+ GetPassNumber(): String
+ CreateStateDuty(SellPrice: double):
+ SetAdditionalInf(carPrice: double, ca
NaturalPersonEntreprenuer1
+ login: String
+ password: String
+ Name: String
+ Surname: String
+ Years: int
+ BirthYear: int
+ Gender: String
+ transactions: ArrayList<Transaction>
- PassNumber: String
- Location: String
- IdentificationCode: int
+ singleTax: Tax
+ singleSocialContribution: Tax
+ CirculationOfMoney: double
+ GetPassNumber(): String
+ SetAdditionalInf(circulationOfMoney:
JuridicalPerson
+ login: String
+ password: String
+ Title: String
+ BirthYear: int
- Location: String
- OSREOUCode: int
+ VAT: ValueAddedTax
+ ExciseTax: ExciseTax
+ IncomeTax: Tax
+ MineralResourcesTax: Tax
+ stateDuty: Tax
+ EcologicalTax: Tax
+ SingleSocialContribution: Tax
+ IncomeNatPerTax: Tax
+ MilitaryTax: Tax
+ RealtyTax: Tax
+ LandTax: Tax
+ TouristicTax: Tax
+ Taxes: ArrayList
+ Income: double
+ PrimeCost: double
+ excise: boolean
+ PercentageofExcise: double
+ mineralUse: boolean
+ PercentageForMines: double
+ Emissions: double
+ CostperTon: double
+ Salaries: double
+ Square: double
+ touristic: boolean
+ NumberOfTourists: int
+ transactions: ArrayList<Transaction>
+ CreateTaxes(): void
+ GetLocation(): String
+ GetOSREOUCode(): int
+ CreateStateDuty(SellPrice: double): v
+ SetAdditionalInf(income: double, prim
NaturalPersonEntrepreneur2
+ juridicalPerson: JuridicalPerson
+ Name: String
+ login: String
+ password: String
+ BirthYear: int
- Location: String
- IdentificationCode: int
+ GetLocation(): String
+ GetIdentificationCode: int
+ CreateTaxes(): void
package sample.User
Users
+ ph: ArrayList<PhysicalPerson>
+ entr1: ArrayList<NaturalPersonEntrepreneur1>
+ entr2: ArrayList<NaturalPersonEntrepreneur2>
+ jr: ArrayList<JuridicalPerson>
+ CreatePhysical(): void
+ CheckExistingPhy(physicalPerson: PhysicalPerson): void
+ CheckPhysical(login: String, password: String): String
+ AddorChangePhysical(number: int, physicalPerson: Physical
+ CreateJuridical(): void
+ CheckExistingJur(juridicalPerson: JuridicalPerson): void
+ CheckJuridical(login: String, password: String): String
+ AddorChangeJuridical(number: int, juridicalPerson: Juridicall
+ CreateEntrepreneur(): void
+ CheckExistingEnt(naturalPersonEntrepreneur1: NaturalPers
+ CheckEntrepreneur(login: String, password: String): String
+ AddorChangeEntrepreneur(number: int, entrepreneur1: Natu
+ CreateEntrepreneur2(): void
+ CheckExistingEnt2(naturalPersonEntrepreneur2: NaturalPer
+ CheckEntrepreneur2(login: String, password: String): String
+ AddorChangeEntrepreneur2(number: int, entrepreneur2: Nat
package sample.Taxes
Transaction
+ Title: String
+ Date: String
+ Amount: Double
+ Approved: boolean
+ Comment: String
+ Tax: Tax
+ DateofApproving: String
+ SetDateofApprove(date: D
<<Interface>> Tax
+ Overpayment: boolean
+ Underpayment: boolean
+ CurrentBalance: doublers
+ calculateTax(): double
+ NewCalculate(): void
+ Pay(sum: double): void
+ NewMounth(): void
+ CheckPayment(): void
«interface» Serialoizable
package sample.Taxes.TaxesPhysicalPerson
LandTaxforPhysicalPerson
LuxuryTax
RealEstateTaxforPhysicalPerson
package sample.Taxes.TaxesNaturalPersonEntrepreneur
SingleTax
package sample.Taxes.TaxesJuridicalPerson
EcologicalTax
ExciseTax
IncomeTaxofJuridicalPerson
IncomeTaxofNaturalPerson
LandTaxforJuridicalPerson
MilitaryFee
RealEstateTaxforJuridicalPerson
RentForUseOfMineralResources
TouristTax
ValueAddedTax
SingleSocialContribution
StateDuty

Main classes are: PhysicalPerson, JuridicalPerson, NaturalPersonEntrepreneur1 and NaturalPersonEntrepreneur2. Class Users uses them to form something like «base» of the program. Almost all main classes have ArrayList of transactions. They also have taxes, type of which is determined while running the application. Class NaturalPersonEntrepreneur2 is an implementation of Adapter pattern, this is why it do not have niether ArrayList of transactions nor taxes.
There exist interface Tax, which includes all needed variables and methods. I did not rewrite variables and methods, because the only diference between them is their constructors and implementation of abstract method *calculateTax().*

Assessment criteria:

Main: the program carries out almost all functions described in the objective and and also those, which where made due to teacher's instructions. To my mind, it also has a correct encapsulation and appropriate use of aggregation, as well as an appropriate use of inheritance and polymorphism in two separate inheritance hierarchies. And the code is organized into packages.

Further: In my opinion, I have fullfilled 5 criteria from the list. I have applicted Addapter pattern (NaturalPersonEnreoreneur2), I am using my own exceptions(ExistingLogion), RTTI and lambda excpressions, graphical user interface is separated from application logic and almost all events are handling manually.

List of submitted working program version:

- Small problems
  https://github.com/OOP-FIIT/oop-2019-uto-16-c-sulaimankhail-olesiamelnychyn/commit/dfbed248a1d0f23e869b77edfce0056d82dc2ef1

- Payment in userpages + History of transactions(with approving by administrator)
  https://github.com/OOP-FIIT/oop-2019-uto-16-c-sulaimankhail-olesiamelnychyn/commit/d873fd53e71fbae8cd2fe5035b11a972237f7add

- Last changes1
  https://github.com/OOP-FIIT/oop-2019-uto-16-c-sulaimankhail-olesiamelnychyn/commit/65c8267780f8908980d422b72ab39388417fcc9f