Maroš Kušnír

SPRÁVA O REALIZÁCII PROJEKTU

Názov:

EZPS

Zámer:

EZPS je Elektronicky zdravotný poisťovací systém, ktorý umožňuje užívateľom okamžitý prístup ku svojim údajom týkajúcich sa zdravotného poistenia v užívateľsky príjemnom grafickom zobrazení. Poistenec bude môcť aktualizovať svoje osobné údaje, prezerať si svoje poistné zmluvy, pozerať sa na ponukové katalógy rôznych zdravotných poisťovní, meniť svoju poisťovňu a poistný program. Okrem toho poistenec sa v tomto systéme bude môcť poistiť špeciálnymi poistnými zmluvami ako je napríklad horské alebo dovolenkové zdravotné poistenie a nim podobné. Zatiaľ čo poisťovacie spoločnosti, štátna zdravotná poisťovňa, ale i súkromné zdravotné poisťovne budú môcť aktualizovať, a taktiež aj rozšíriť ich ponukový katalóg poistných programov. Systém im poskytne prehľad o poistencoch a bude ich kategorizovať podľa výšky obvodu a deliť poistencov na ľudí v predproduktívnom veku, produktívnom veku a poproduktívnom veku, čím poskytne poisťovne detailnejší prehľad o poistencoch. Systém bude môcť ovplyvňovať aj samotný štát, ktorý bude špecifikovať minimálny odvod do zdravotnej poisťovne. A taktiež bude definovať hranice pre progresívne odvody do zdravotného systému, čiže ak človek bude zarábať nad určitú sumu štát je schopný zadefinovať progresívne zdanenie. Okrem toho štát bude mať prehľad o množstve ľudí v zdravotnom systéme a taktiež aj o množstve ľudí, za ktorých platí štát zdravotné odvody.

Zoznam hlavných verzií:

996a9ee *comment*: final push with java

38af9d8 *comment* lambda functions and exception methods

2bab825 *comment*: imported for eclipse

ca2be2f *comment* added working full fledge polymorphism

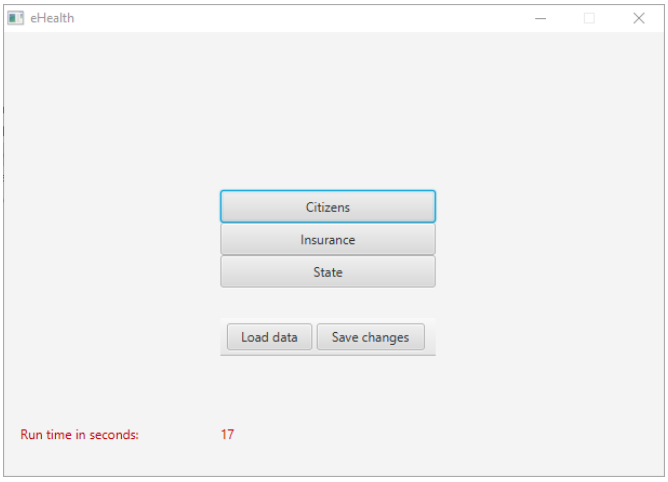
Zoznam všetkých verzií je v súbore log.svg na githube

Maroš Kušnír

Tutoriál

Aplikácia sa spúšťa pustením main metódy v Main triede, ktorá je v balíku startingPoint.

Po spustený sa zobrazí grafické rozhranie WelcomeScene, ktorý má niekoľko tlačidiel.



Prvé tlačidlo vstúpi do menu pre akcie aktéra osôb.

Druhé tlačidlo vstúpi do menu pre akcie aktéra poisťovne.

Tretie tlačidlo vstúpi do menu pre akcie štátu.

Load data tlačidlo načíta uložené dáta do programu.

Save changes tlačidlo uloží zmeny do programu ale prepíše uložené dáta, POZOR

Run time in seconds meria čas neaktivity na tejto scéne sa napočíta do 30 program sa vypne.

V ďalších scénach sa taktiež používajú Tlačidlá na prechody.

No niektoré ďalšie scény aj prijímajú input. Tam je postup jasný ak je tam tlačidlo refresh tak ho treba stlačiť na to aby sa vypísali informácie do TextArea. Potom podľa vypísaných poisťovní alebo osôb vyber ich index ktorý je pri nich, ak s nimi budete chcieť robiť ďalšie zmeny

Zhrnutie:

Program umožňuje registrovať osoby do systému, registrovať ďalšie poisťovne do systému. Umožňuje meniť osobám poisťovne spolu s poistným programom. Umožňuje štátu nadstaviť poplatky pre tri kategórie zarábajúcich ľudí (progresívne zdanenie). Taktiež umožňuje načítať dáta a uložiť zmeny. Okrem toho poskytuje rôzne druhy výpisov pre aktéra poisťovní. Napríklad výpis poistených ľudí iba z vybranej poisťovne alebo výpis iba ponukových katalógov z vybranej poisťovne. A v neposlednom rade poskytuje možnosť rozširovať poisťovací katalóg o nové poistné programy.

Maroš Kušnír

Diagramy:

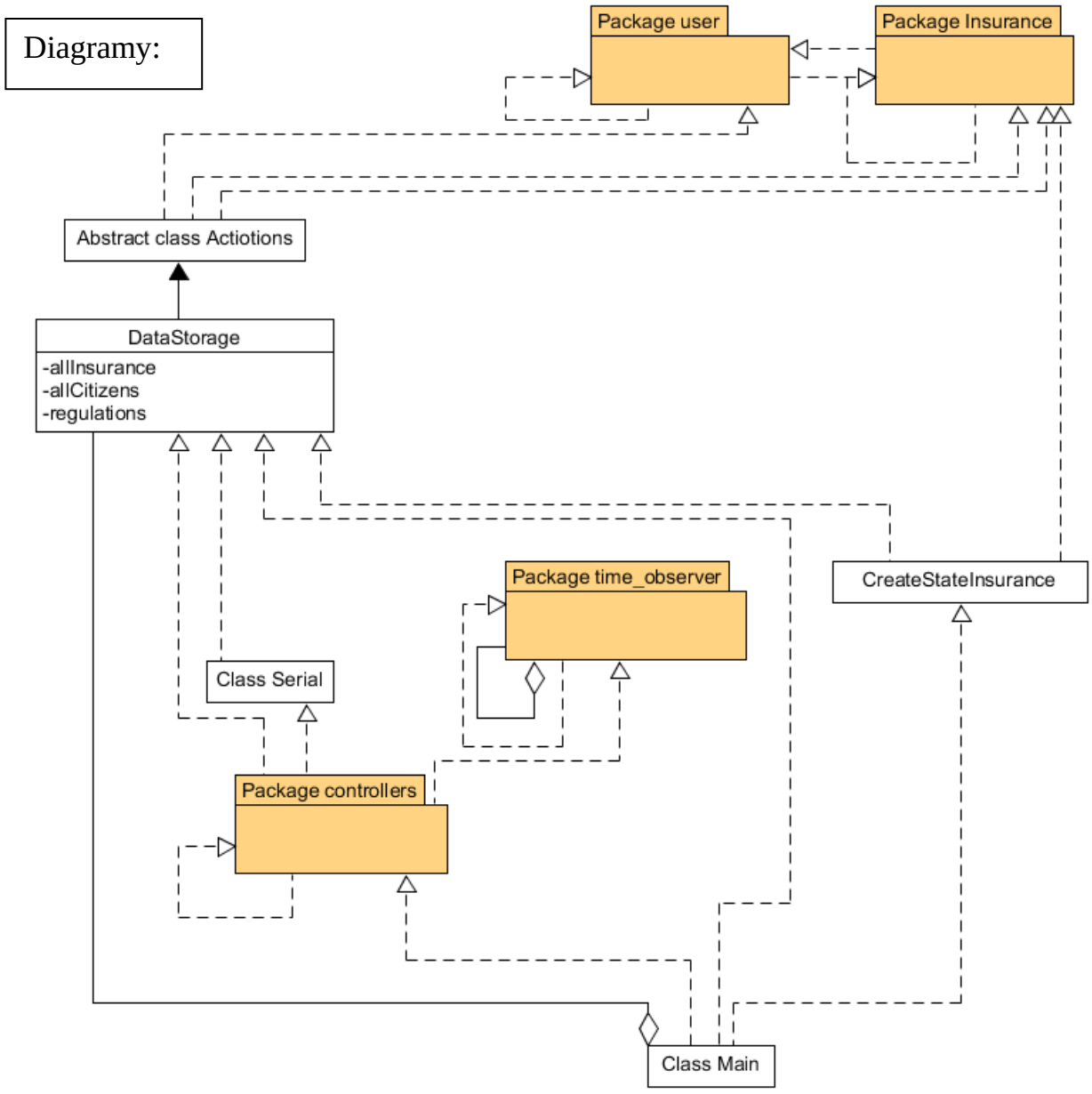
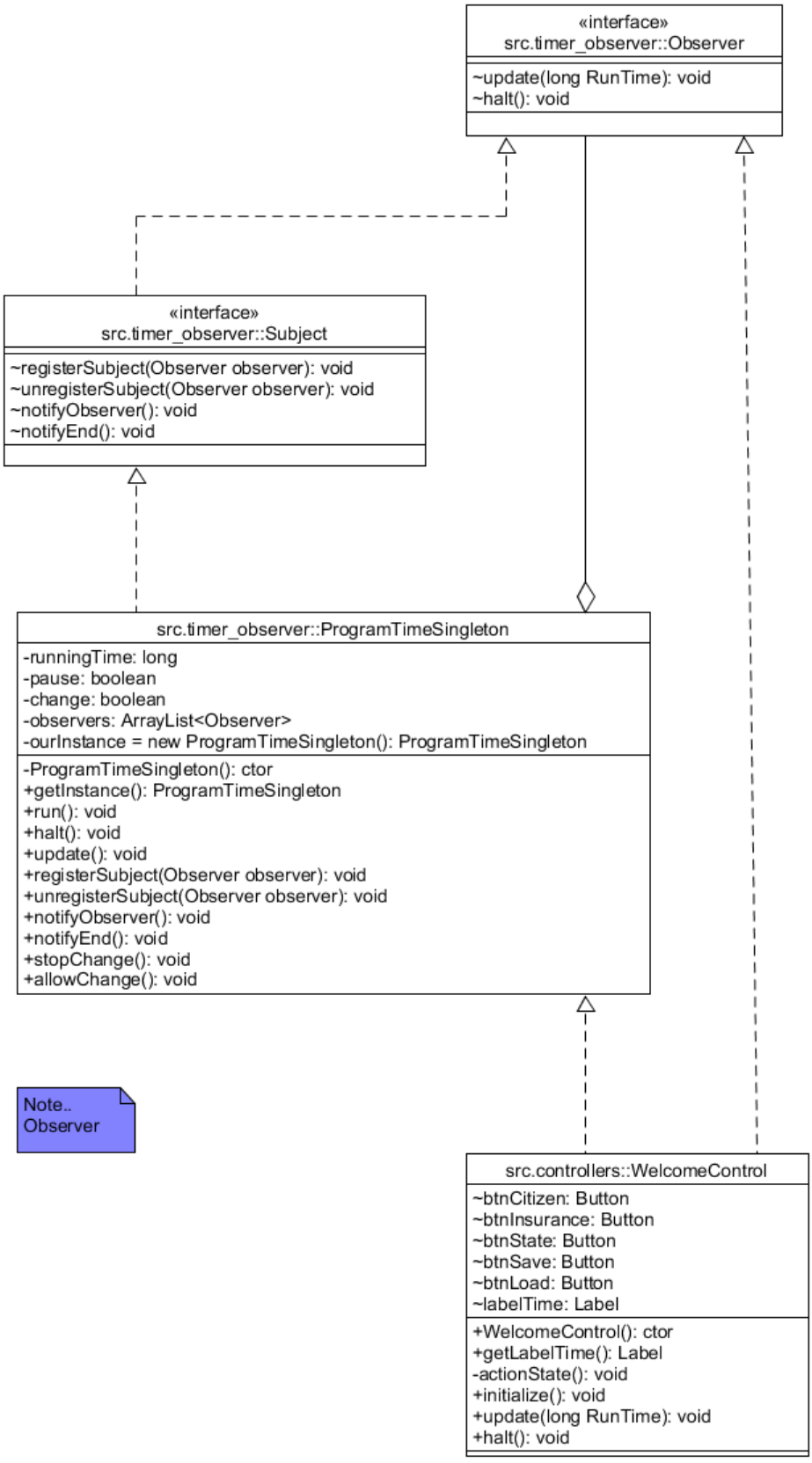


Diagram celého rozloženia programu z Main triedy sa spustí program, spustí metódu na vytvorenie DataStorage, kde sú dáta a logika programu. Potom spustí metódu na spustenie gui, kde sa bude predávať stage a state. Okrem toho na začiatku inicializuje a vytvorí štátnu poisťovňu s predprogramovanými hodnotami. Controlleri spúšťajú akcie vykonávané v Action triede nad údajmi z DataStrorage. Actions vykonávajú zmenu nad usermi a poisťovňami (nad aktérmi). Poisťovne a useri (packege user a package Insurance) sa môžu navzájom ovplyvňovať a meniť svoje hodnoty prostredníctvom napríklad Visitora, ktorý je rozpísaný nižšie vo vedľajších kritériách no diagram je v tejto sekcii. Okrem toho v Main vytvorí aj časovač, ktorý je singleton a beží na vlastnom threade po celú dĺžku programu. Okrem toho časovač je zároveň observerom, ktorý notifikuje o zmene času.

Maroš Kušnír



Tento graf znázorňuje implementáciu návrhového vzoru observera v kóde.

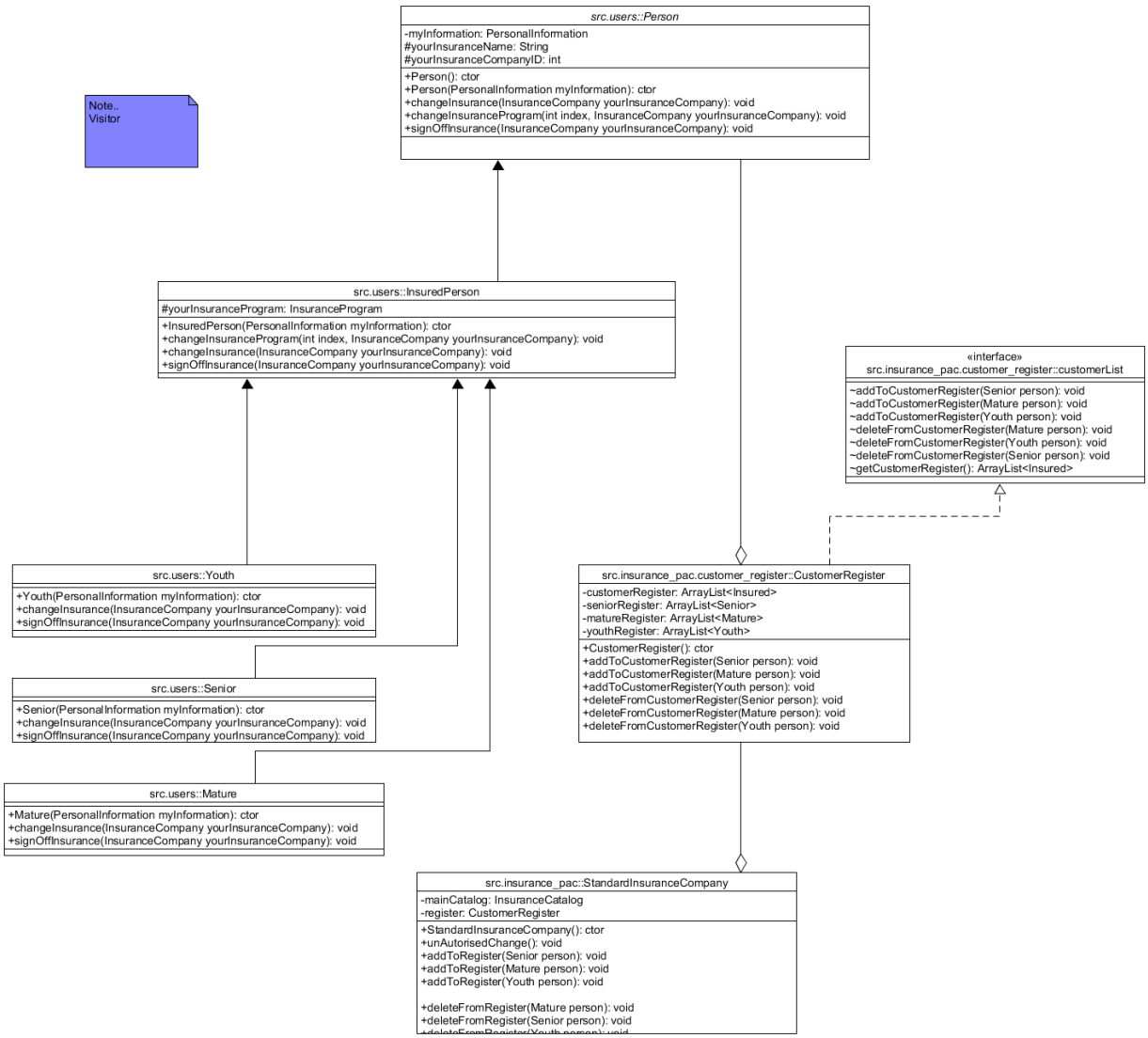
Samotný kód je popísaný vo vedľajších kritériách.

ProgramTimeSingleton beží na vlastnom threade. Každou sekundou notifikuje napríklad WelcomeController. Vo WelcomeControlleri sa notifikáciou zmení hodnota čísla v Labelu.

Ak program napočíta do 30 program skončí, teda pokiaľ, používateľ programu neprepne na inú scénu. Tento časovač inaktivity je iba na WelcomeScene.

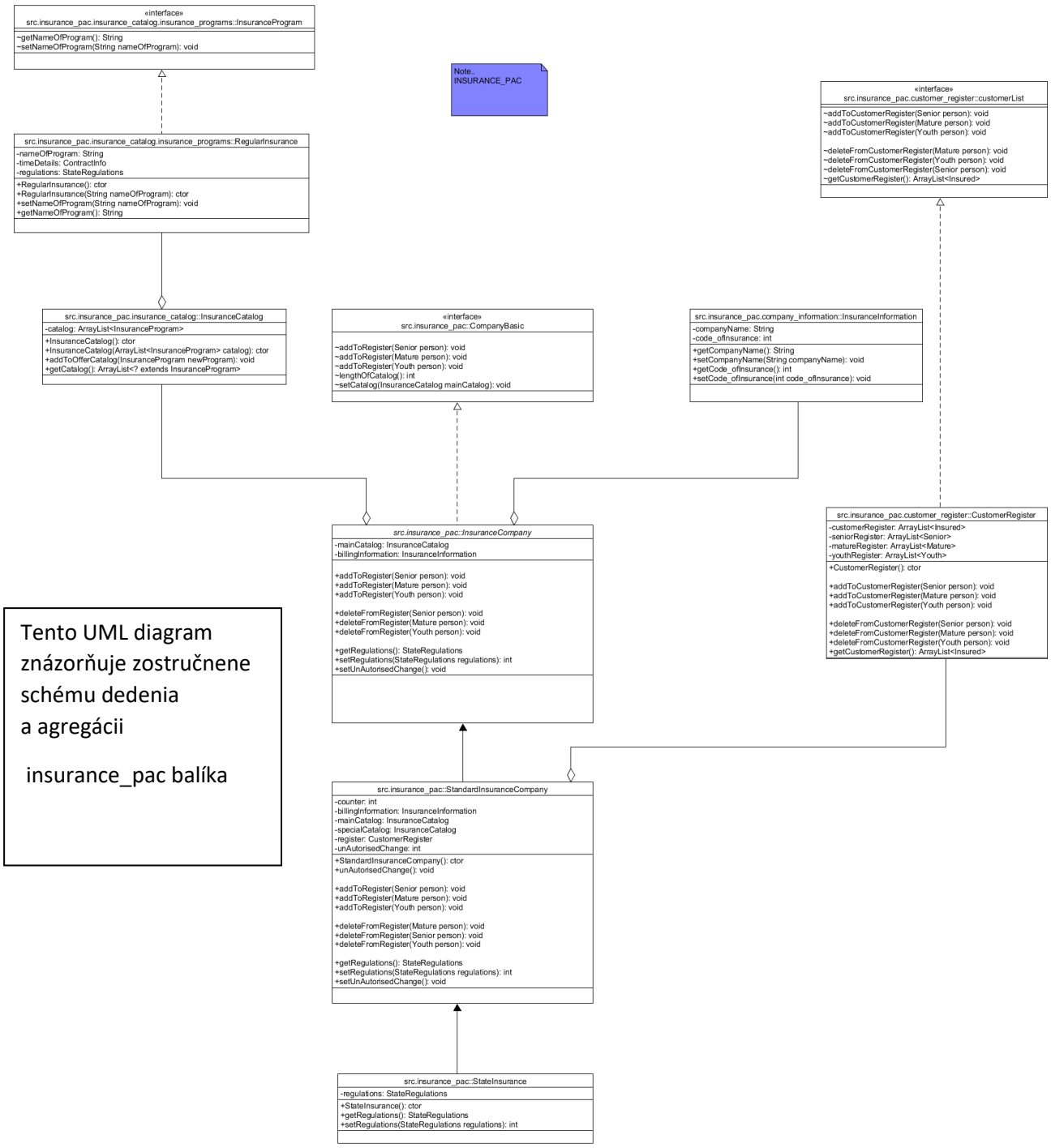
Observerovi sa venujem vo vedľajších kritériách

Maroš Kušnír



Nasledujúci diagram ukazuje štruktúru kde sa vyskytuje dizajnový vzor Visitor. Implementácia a prevedenie v kóde je vo vedľajších kritériach.

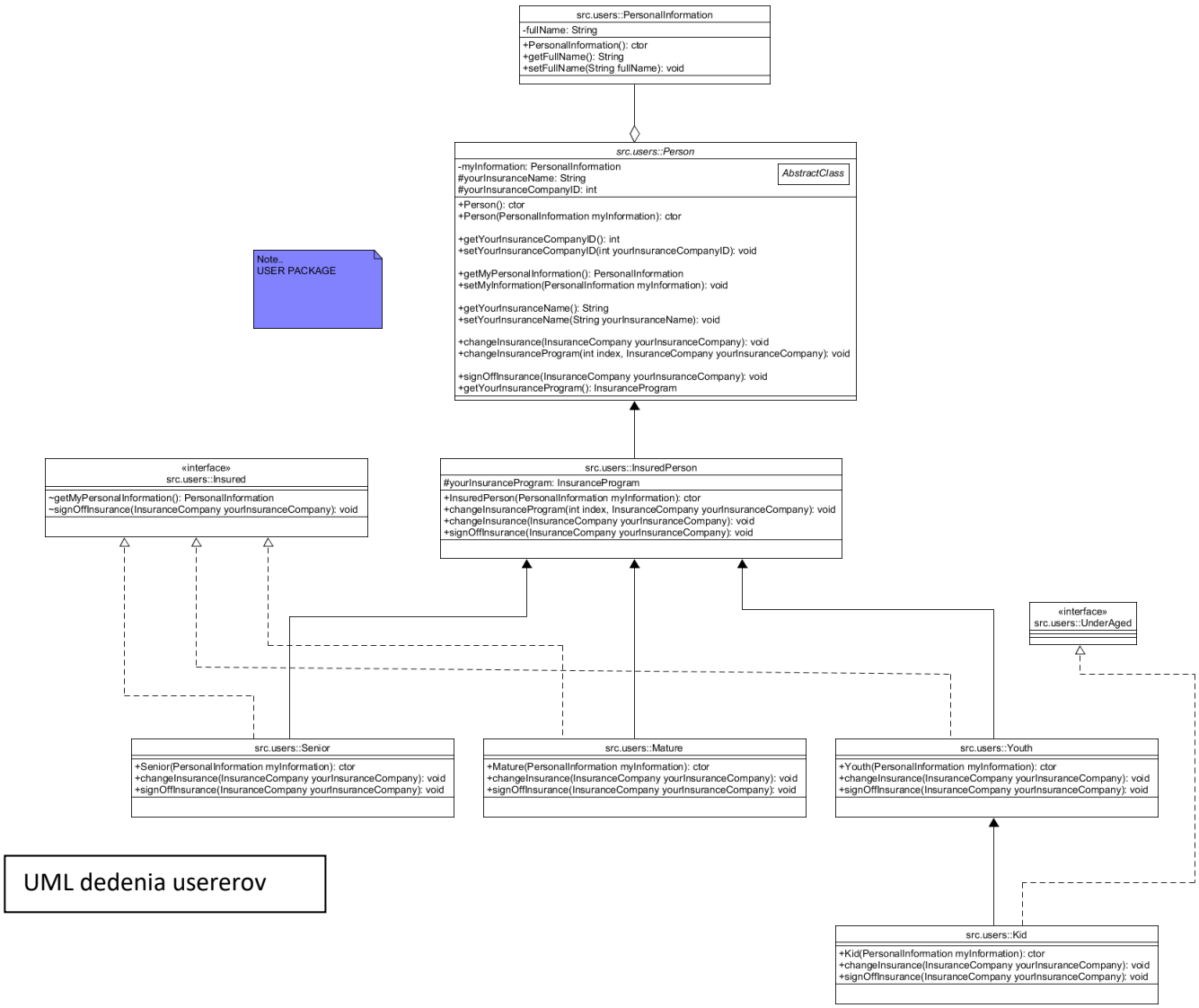
Maroš Kušnír



Tento UML diagram znázorňuje zostručnene schému dedenia a agregácii

insurance_pac balíka

Maroš Kušnír



UML dedenia usererov

Maroš Kušnír

Hlavné kritéria:

POLYMORFIZMUS, DEDENIE, PREKONANIE VLASTNÝCH METÓD A ZMYSLUPLNÉ DEDENIE

Pri realizácii polymorfizmu dochádza k prekonaniu metód, a tým pádom aj k zmysluplnému dedeniu.

Tak ako podmienky projektu kážu mám dva polymorfizmy v dvoch oddelených hierarchiách dedenia.

Prvá sa vyskytuje pri v **balíku user** medzi **class Youth a class Kid v** spomínanom balíku pri metóde:

```
public void changeInsurance(InsuranceCompany yourInsuranceCompany) {…}
```

```
public class Youth extends InsuredPerson implements Insured{

    /**
     * Class constructor
     * @param myInformation
     * myInformation PersonalInformation as an parameter
     */
    public Youth(PersonalInformation myInformation) { super(myInformation); }

    /**
     * Method that enables user to change Insurance
     * @param yourInsuranceCompany
     * yourInsuranceCompany InsuranceCompany is a parameter,
     * it uses InsuranceCompany as a visitor
     * it takes care of registration of Youth member
     */
    @Override
    public void changeInsurance(InsuranceCompany yourInsuranceCompany) {

        yourInsuranceCompany.addToRegister( person: this);
        yourInsuranceName =  yourInsuranceCompany.getBillingInformation().getCompanyName();
        yourInsuranceCompanyID =  yourInsuranceCompany.getBillingInformation().getCode_ofInsurance();

    }
```

```java
public class Kid extends Youth implements UnderAged{

    /**
     * Class constructor
     * @param myInformation
     * myInformation PersonalInformation as an parameter
     */
    public Kid(PersonalInformation myInformation) {
        super(myInformation);
    }

    /**
     * Method that enables user to change Insurance
     * BUT class kid is restricted so it can be set only to state insurance_pac
     * @param yourInsuranceCompany
     * yourInsuranceCompany InsuranceCompany is a parameter,
     * it uses InsuranceCompany as a visitor
     * it takes care of registration of Mature member
     */
    @Override
    public void changeInsurance(InsuranceCompany yourInsuranceCompany) {
        if(yourInsuranceCompany.getBillingInformation().getCode_ofInsurance()==0){
            //System.out.println("Hura");
            yourInsuranceCompany.addToRegister( person: this);
            yourInsuranceName =  yourInsuranceCompany.getBillingInformation().getCompanyName();
            yourInsuranceCompanyID =  yourInsuranceCompany.getBillingInformation().getCode_ofInsurance();
            return;
        }

        yourInsuranceCompany.setUnAutorisedChange();
    }///polymorphism
```

Kid dedí od Youth class. Kid Prekonáva metódu changeInsurance a tým vytvára znak polymorfizmus. No či je to polymorfizmus závisí od implementácii. Implementácia tohto polymorfizmu sa odohráva v classe Action v metóde: confirmChange nasledovným spôsobom. Z ArrayListu (kde môžu byť objekty typu Person) sa vyberie jeden podľa zadaného indexu. S tým, že to môže byť Mature, Senior, Youth a Kid objekt. No to nevieme. A na tomto vybranom objekte vykonáme akciu changeInsurance. To čo sa stane závisí od typu.

```java
person.changeInsurance(insuranceCompany);
```

Napríklad ak je to Youth, tak sa mu zmení poisťovňa, to znamená poisťovňa ho zaregistruje. Zmení sa mu názov poisťovne vo vnútri objektu a aj index poisťovne v aktuálnom objekte sa zmení. No ak by to bol objekt typu Kid stane sa to iba ak je to štátna poisťovňa, inak nevykoná spomínané akcie ale iba upovedomí poisťovňu o neautorizovanej zmene prostredníctvom metody .setUnAuthorisedChange().

Maroš Kušnír

Druhý polymorfizmus sa **vyskytuje v balíku insurance_pac** medzi class **StandardInsuranceCompany** a class **StateInsurance**.

```
public class StateInsurance extends StandardInsuranceCompany
```

//nasledujúca metóda je v StateInsurance class

```
@Override
public int setRegulations(StateRegulations regulations) {

    this.regulations = regulations;
    return 1;
}
```

```
public class StandardInsuranceCompany extends InsuranceCompany
```

//nasledujúca metóda je v Standard Insurance class

```
@Override
public int setRegulations(StateRegulations regulations) {
    this.unAutorisedChange();
    return 0;
}
```

Implementácia sa tiež odohráva v triede Actions nasledovne:

```
for (InsuranceCompany company: controller.getState().getAllInsurance()){

    value = company.setRegulations(controller.getState().getRegulations());

}
```

AGREGÁCIA

Vyskytuje sa skoro všade, na príklad:

```
1
2 package src.insurance_pac;
3 ...
4 public class StateInsurance extends StandardInsuranceCompany {
5
6     private StateRegulations regulations;
7 ...
```

```
1 package src.users;
2 ...
3 public abstract class Person implements Serializable {
4
5     private PersonalInformation myInformation;
6 ...
7 }
```

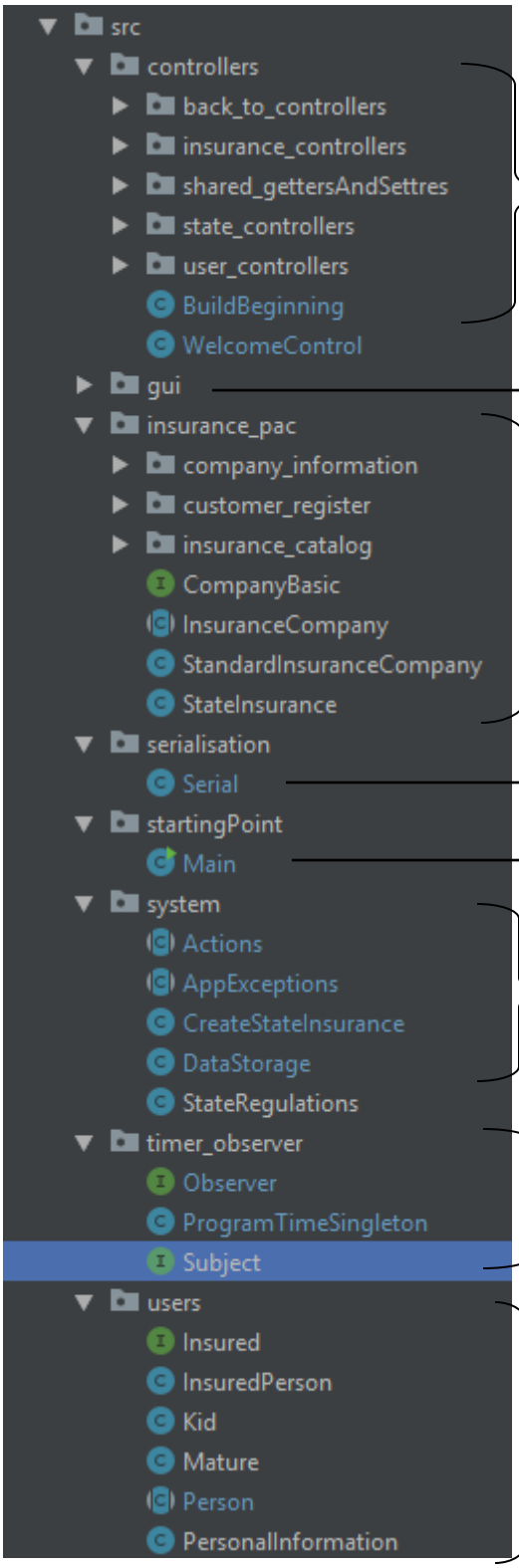
```
1 package src.insurance_pac.customer_register;
2 ...
3 public class CustomerRegister implements customerList, Serializable {
4
5     private ArrayList <Insured> customerRegister;
6     private ArrayList <Senior>  seniorRegister;
7     private ArrayList <Mature>  matureRegister;
8     private ArrayList <Youth>  youthRegister;
```

ZAPUZDRENIE

Naprieč celým projektom, veľká premenné v triedach sú buď private alebo protected a sú dostupné prostredníctvom getterov a setterov. Každá trieda má metódy, ktoré s ňou operujú.

Maroš Kušnír

ORGANIZOVANÝ DO BALÍKOV



Balík controllers obsahuje všetky controller.

Samotné controller-i sú rozdelené poďla toho či vykonávajú akcie and userom, poisťovňou alebbo štátom

Potom sú tam ešte balíky, kt. obsahujú zdielanú funkcionalitu s ostatnými controlermi

Obsahuje scenes v XML kóde

Všetky dedenia poisťovní a všetko čo sa jej týka je ďalej vo vlastných balíkov, napríklad cutomer_register, (základné informácie) company_information, (poistný katalóg) insurance_catalog.

Serializacia a deserializacia

TU SA SPÚŠŤA PROGRAM

Dáta a akcie vykonávané nad nimi. Táto časť berie input od controllerov

Vlastný časovač, ktorý je časť observera a je to singleton, má jedinečnú funkcionalitu odlišnú od ostatných a vykonáva sa ešte na vlastnom threade

Triedy a kategorizácia ľudí

Aplikácia ďalších kritérií:

OŠETRENIE MIMORIADNYCH STAVOV PROSTREDNÍCTVOM VLASTNÝCH VÝNIMIEK

Class AppException v packegu system:

```
1 public abstract class AppExceptions {
2
3     class NotInRangeException extends Exception{
4
5         private String message;
6
7         public NotInRangeException(String message) {
8             super(message);
9             this.message=message;
10        }
11
12        public void alertSend(InsuranceModify controller){
13            controller.errorMessage(""+this.getMessage());
14        }
15    }
16
17    class NotATypeException extends Exception{
18
19
20        public NotATypeException(String message) {
21            super(message);
22        }
23
24        public void alertSend(UserAdd controller){
25            controller.errorMessage(""+this.getMessage());
26        }
27    }
28
29    class ParsingException extends Exception{
30
31
32        public ParsingException(String message) {
33            super(message);
34        }
35
36        public void alertSend(InsuranceModify controller){
37            controller.errorMessage(""+this.getMessage());
38        }
39
40        public void alertSend(InsuranceCatalogShow controller){
41            controller.errorMessage(""+this.getMessage());
42        }
43    }
44
45 }
```

Kód nad týmto textom ukazuje triedu AppExceptions, v ktorej sú spravené

vnorené triedy pre jednotlivé ***vlastné výnimky***. Každá z týchto vnorených tried má ***vlastnú metódu***, ktorá má v parametre príslušný controller, z ktorého dokáže spusiť (alert window) okno s upozornením.

```
 private void testProgram(InsuranceModify controller) throws
NotInRangeException,ParsingException {

1         int value =0;
2         RegularInsurance program = new RegularInsurance();
3         program.setNameOfProgram(""+controller.getTxtProgramName().getText());
4
5         try {
6             value = Integer.parseInt(controller.getIndex().getText());
7         }catch (Exception e){
8
9             ParsingException missing =
10             new ParsingException("Input is not a whole number \n");
11
12            missing.alertSend(controller);
13            throw missing;
14        }
15
16
17        if((controller.getState().getAllInsurance().size() >= value) && (value >= 0)
18 ){
19              controller.getState().getOneInsurance(value).
20              getCatalog().addToOfferCatalog(program);//
21            return;
22        }
23        NotInRangeException missing =
24         new NotInRangeException("That insurance_pac does not exist \n");
          missing.alertSend(controller);
          throw missing;
 }
```

Kód nad týmto textom ukazuje už využitie vlastných výnimiek v praxe.

Maroš Kušnír

POUŽITIE VHNIEZDENÝCH TRIED A ROZHRANÍ

Použitie „vnorených" (nested classes) tried som ukázal v ukážke kódu kde sa zaoberám bodom ohľadom vlastných výnimiek. Použil som totiž triedu, do ktorej som umiestnil všetky vlastné výnimky.

Vhniezdené rozhranie som implementoval na mieste:
\src\insurance_pac\CompanyBasic.java

```
1  public interface CompanyBasic {
2
3      InsuranceCatalog getCatalog();
4      InsuranceInformation getBillingInformation();
5      CustomerRegister getRegister();
6
7      void setBillingInformation(InsuranceInformation billingInformation);
8      void addToRegister(Senior person);
9      void addToRegister(Mature person);
10     void addToRegister(Youth person);
11     int lengthOfCatalog();
12     void setCatalog(InsuranceCatalog mainCatalog);
13
14
15     interface CompanyDeletions{
16
17         void deleteFromRegister(Senior person);
18         void deleteFromRegister(Mature person);
19         void deleteFromRegister(Youth person);
20     }
21
22 }
```

Maroš Kušnír

POUŽITIE LAMBDA VÝRAZOV, REFERENCIÍ NA METÓDY (METHOD REFERENCES)

Ukážka z kódu:

Lambda a referencia na metodu:

Miesto: \src\system\Actions.java

Methoda: *printCatalog*

```
controller.getState().getOneInsurance(value).getCatalog().getCatalo
g()
        .forEach(e ->
controller.getDisplaySecond().appendText(e.getNameOfProgram() +
"\n"));
```

Lambda a referencia na metodu:

Miesto: \src\system\Actions.java

Methoda: *insuranceExistCheck*

```
List<String> names =
controller.getState().getAllInsurance().stream().map(c ->
c.getBillingInformation().getCompanyName())
.collect(Collectors.toList());

boolean contains = names.stream().anyMatch(inputName::equals);
```

Lambda a referencia na metodu:

Miesto: \src\system\Actions.java

Methoda: *userExistCheck*

```
List<String> names =
controller.getState().getAllCitizens().stream().map(c ->
c.getMyPersonalInformation().getFullName()).collect(Collectors.toLi
st());
boolean contains = names.stream().anyMatch(inputName::equals);
```

Maroš Kušnír

EXPLICITNÉ POUŽITIE VIACNIŤOVOSTI (MULTITHREADING)

Class ProgramTimeSingleton extendujem Thread, ktorý potom spúšťam.

```
1  package src.timer_observer;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5
6  public class ProgramTimeSingleton extends  Thread implements Subject,
7  Serializable {
8
9  ...
10
11     public void run(){
12         pause=false;
13         change=true;
14         while(!pause){
15             //System.out.println("running");
16             if (change){
17                 this.update();
18                 if(runningTime==30L){
19                     this.notifyEnd();
20                 }
21             }
22             try {
23                 Thread.sleep(1000);
24             } catch (InterruptedException e) { }
25         }
26
27     }
28
29 ...
   } //... tri bodky reprezentuju preskoceny kod
```

Maroš Kušnír

EXPLICITNÉ POUŽITIE RTTI

```
if(person instanceof UnderAged){
    return;
}
```

Zisťujem či na či na osobu bolo použité rozhranie neplnoletý (UnderAged). Ak hej tak metóda skončí returnom. Keďže kategória neplnoletých sa môže použiť na ľubovoľnú triedu alebo podtriedu userov a zároveň neplnoletý ma z právneho hľadiska, mňa ako poisťovací systém zaujíma. Preto považujem za adekvátne použitie RTTI a nie polymorfizmu keďže tu ma zaujíma iba, to či dotyčný je alebo nie je plnoletý.

Maroš Kušnír

POSKYTNUTIE GRAFICKÉHO POUŽÍVATEĽSKÉHO ROZHRANIA ODDELENE OD APLIKAČNEJ LOGIKY A S ASPOŇ ČASŤOU SPRACOVATEĽOV UDALOSTI (HANDLERS) VYTVORENOU MANUÁLNE

Controllery, logika a sceny (XML) sú oddelené v balíkoch. Logika niej e súčasťou controller-ov.

```
1   public void initialize(){
 2
 3         btnAddPerson.setOnAction(e -> {
 4             try {
 5                 this.actionUser();
 6             } catch (Exception e1) {
 7                 e1.printStackTrace();
 8             }
 9         });
10
11         btnList.setOnAction(e -> {
12             try {
13                 this.actionDisplay();
14             } catch (Exception e1) {
15                 e1.printStackTrace();
16             }
17         });
18
19         btnChange.setOnAction(e -> {
20             try {
21                 this.actionChange();
22             } catch (Exception e1) {
23                 e1.printStackTrace();
24             }
25         });
26
27         btnHome.setOnAction(e -> {
28             try {
29                 this.actionMenu();
30             } catch (Exception e1) {
31                 e1.printStackTrace();
32             }
33         });
34
35
36         btnCheck.setOnAction(e -> {
37             try {
38                 this.actionCheck();
39             } catch (Exception e1) {
40                 e1.printStackTrace();
41             }
42         });
43     }
```

Kód nad týmto je kód z programu, kde sú manuálne spravené handlers.

Handlers sú spravené manuálne napríklad na mieste:

Maroš Kušnír

src\src\controllers\user_controllers

hoci jaký file z user_controllers balíka

Maroš Kušnír

POUŽITIE NÁVRHOVÝCH VZOROV

Použil som tri návrhové vzory: observer, visitor a singleton.

Observer:

Rozhrania:

Pakage timer_observer

```
1 public interface Observer {
2
3     void update(long RunTime);
4     void halt();
5 }
```

Pakage timer_observer

```
1 public interface Subject {
2
3     void registerSubject   (Observer observer);
4     void unregisterSubject (Observer observer);
5     void notifyObserver();
6     void notifyEnd();
7
8 }
```

Triedy s implementovanými rozhraniami:

Balík timer_observer

```
1  ...
 2 public class ProgramTimeSingleton extends  Thread implements Subject,
 3 Serializable {
 4
 5     private long runningTime;
 6     private boolean pause;
 7     private boolean change;
 8     private ArrayList<Observer> observers;
 9
10 ...
11
12 ...
13     public void run(){
14         pause=false;
15         change=true;
16         while(!pause){
17             //System.out.println("running");
18             if (change){
19                 this.update();
20                 if(runningTime==30L){
21                     this.notifyEnd();
22                 }
```

```
23              }
24              try {
25                  Thread.sleep(1000);
26              } catch (InterruptedException e) { }
27          }
28
29      }
30 ...
31      public void update() {
32          System.out.println("" + runningTime++);
33          this.getInstance().notifyObserver();
34      }
35      @Override
36      public void registerSubject(Observer observer) {
37          observers.add(observer);
38          //System.out.println("hura registroval");
39      }
40      @Override
41      public void unregisterSubject(Observer observer) {
42          int inde = observers.indexOf(observer);
43          observers.remove(inde);
44      }
45
46      @Override
47      public void notifyObserver() {
48          //System.out.println("Notified one");
49          for (Observer observer: observers) {
50              observer.update(runningTime);
51              //System.out.println("Notified");
52          }
53      }
54
55      @Override
56      public void notifyEnd() {
57          for (Observer observer: observers) {
58              observer.halt();
59          }
60          this.halt();
61      }
62 ...
63
   }
```

Notifikovaná trieda:

Balík controllers,

Trieda: `WelcomeControl`

```
1  ...
 2 public class WelcomeControl extends Shared implements Observer{
```

```
 3     ...
 4     ...
 5     ...
 6      public WelcomeControl() {
 7          ProgramTimeSingleton.getInstance().registerSubject(this);
 8          ProgramTimeSingleton.getInstance().allowChange();
 9      }
10
11      @Override
12      public void update(long RunTime) {
13          Platform.runLater(()->{
14              this.getLabelTime().setText(""+ RunTime);
15              //System.out.println("" + id+ ": " +time);
16          });
17      }
18
19      @Override
20      public void halt(){
21          Platform.runLater(()->{
22              this.getStage().close();
23          });
24      }
25
26 }
```

Visitor:

Rozhrania

Balík:

\src\users\Insured.java

```
1 public interface Insured {
2     PersonalInformation getMyPersonalInformation();
3     void signOffInsurance(InsuranceCompany yourInsuranceCompany);
4 }
```

Balík:

\src\insurance_pac\customer_register\customerList.java

```
1  public interface customerList {
 2
 3      void addToCustomerRegister(Senior person);
 4      void addToCustomerRegister(Mature person);
 5      void addToCustomerRegister(Youth person);
 6
 7      void deleteFromCustomerRegister(Mature person);
 8      void deleteFromCustomerRegister(Youth person);
 9      void deleteFromCustomerRegister(Senior person);
10
11      ArrayList<Insured> getCustomerRegister();
12 }
```

Maroš Kušnír

Triedy s implementovanými rozhraniami:

Visitor

Balík: \src\users

```
1  ...
 2 public class Senior extends InsuredPerson implements Insured {
 3
 4 ...
 5     @Override
 6     public void changeInsurance(InsuranceCompany yourInsuranceCompany) {
 7
 8         yourInsuranceCompany.addToRegister(this);
 9         ...
10     }
11 ...
12 }
```

```
1  ...
 2 public class Mature extends InsuredPerson implements Insured {
 3
 4 ...
 5     @Override
 6     public void changeInsurance(InsuranceCompany yourInsuranceCompany) {
 7
 8         yourInsuranceCompany.addToRegister(this);
 9         ...
10     }
11 ...
12 }
```

Objekty sú odovzdané ešte do medzi kroku do poisťovne kde sú odovzdané nasledovným ďalej do metódy visitora. Dôvod je ten, že Person by nemal mať priamy prístup do registra poisťovne.

Balík: \src\insurance_pac

```
 1 ...
 2 public class StandardInsuranceCompany extends InsuranceCompany implements
 3 Serializable {
 4 ...
 5
 6     @Override
 7     public void addToRegister(Mature person) {
 8         ...
 9         register.addToCustomerRegister(person);
10        ...
11     }
12 @Override
13     public void addToRegister(Senior person) {
14         ...
15         register.addToCustomerRegister(person);
```

```
16          ...
17      }
18 ...
   }
```

Visitor:

Balík: src\insurance_pac\customer_register

```
1  public class CustomerRegister implements customerList, Serializable {
 2
 3      private ArrayList <Insured> customerRegister;
 4      private ArrayList <Senior>  seniorRegister;
 5      private ArrayList <Mature>  matureRegister;
 6      private ArrayList <Youth>  youthRegister;
 7
 8 ..
 9
10 ///
11      @Override
12      public void addToCustomerRegister(Senior person) {
13          customerRegister.add(person);
14          seniorRegister.add(person);
15      }
16
17      @Override
18      public void addToCustomerRegister(Mature person) {
19          customerRegister.add(person);
20          matureRegister.add(person);
21      }
22   ...
23 }
```

Zhrnutie visitora používam na zmenu a registráciu do poisťovne a taktiež na jej odhlásenie. Observera na meranie času inaktivity v hlavnom menu, vo WelcomeScene.