**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**
**SLOVENSKEJ TECHNICKEJ UNIVERZITY V BRATISLAVE**

# Internetbanking

Objektovo-orientované programovanie

Zadanie 3

Erik Matovič, ID: 98347

Cvičenie: Streda 18:00
Cvičiaci: Ing. Waheedullah Sulaiman Khail, M.Sc. Rok:2019/2020

## Obsah:


Úvod ... 2
Zadanie – Dôveryhodný softvér ... 3
Zámer projektu – Internetbanking ... 4
Diagram tried v jazyku UML ... 5
Vysvetlenie tried a ich vzťahov ... 5
Splnené hlavné kritéria hodnotenia ... 7
Program v súlade so zadaním ... 7
Program v súlade so zámerom projektu ... 7
Dedenie ... 7
Polymorfizmus v aspoň 2 oddelených hierarchiách dedenia ... 8
Rozhrania ... 9
Zapuzdrenie ... 9
Agregácia ... 9
Oddelenie aplikačnej logiky od používateľského rozhrania ... 10
Kód vhodne organizovaný do balíkov ... 10
Prehľadná dokumentácia so všetkými položkami podľa opisu ... 10
Splnené ďalšie kritéria hodnotenia ... 12
Návrhové vzory(Observer, Builder a Factory) ... 12
Ošetrenie mimoriadnych stavov prostredníctvom vlastných výnimiek ... 14
Použitie implicitnej implementácie metód v rozhraniach ... 15
Poskytnutie grafického používateľského rozhrania oddelene od aplikačnej logiky a s aspoň časťou spracovateľov udalostí (handlers) vytvorenou manuálne ... 16
Použitie vhniezdených tried a rozhraní ... 16
Lambda výrazy ... 17
Zoznam hlavných verzií programu odovzdaných do GitHub classroomu ... 18
Javadoc dokumentácia ... 19

## Úvod

Program je nutné spúšťať cez *Main.java* vo verzii jdk1.8.0_241.

Pri spustení programu je nutné zadať username(a/b) a password(a - 123/b - 456). Po overení prihlasovacích údajov(po stlačení tlačidlá Log In) sú dostupné tlačidlá Accout slúžiace pre výpis informácií(zostatok na účte, meno účtu a dostupné karty s informáciami o nich), Transfer money slúžiace pre prenos peňazí, Log Out slúžiace na odhlásenie.

Po stlačení Transfer money užívateľ môže zadať sumu na prevod peňazí(pri desatinnom čísle je nutné zapísať s bodkou, nie s čiarkou, napr. 12.5 a NIE 12,5) a po stlačení tlačidla Send money vyskočí overovacie okno pre potvrdenie transakcie, po stlačení Yes nastane prevod peňazí z jedného účtu na druhý, vyskočí okno s výpisom, že prevod bol úspešný. Ďalej je dostupné okno Go back to user menu, ktoré užívateľa vráti späť, ale neodhlási ho, čiže si užívateľ môže vybrať, či si chce po stlačení tlačidla Account zobraziť informácie o účte alebo iné možnosti.

## Zadanie – Dôveryhodný softvér

Všadeprítomnosť softvéru zviditeľňuje dôležitosť jeho dôveryhodnosti. Tá sa prejavuje cez ochranu, spoľahlivosť, dostupnosť, odolnosť a bezpečnosť. Ochrana predpokladá manažment rizík, spoľahlivosť a dostupnosť sa odzrkadľujú najmä v očakávanom správaní systému v priebehu času, odolnosť znamená udržanie funkčnosti systému za sťažených podmienok, kým bezpečnosť predstavuje cielené aktivity proti zámerným útokom.

Dôveryhodnosť možno vnímať ako mimofuknčný (nefunkcionálny) aspekt, ale môže byť realizovaná aj vyhradenými riadiacimi prvkami. Možné aplikácie zahŕňajú systémy na správu citlivých údajov, budov, dopravy, organizácií, spojení atď.

V programe, ktorý budete tvoriť podľa vášho zámeru, nebude potrebné riešiť sieťové záležitosti alebo súčasné používanie viacerými používateľmi (napr. reálnu komunikáciu viacerých používateľov cez sieť). Tieto činnosti stačí napodobniť (napr. dá sa prihlásiť ako jeden používateľ, a potom ako druhý). Rovnako nie je potrebné riešiť ani bezpečnosť. Sústreďte sa na objektovo-orientovanú modularizáciu.

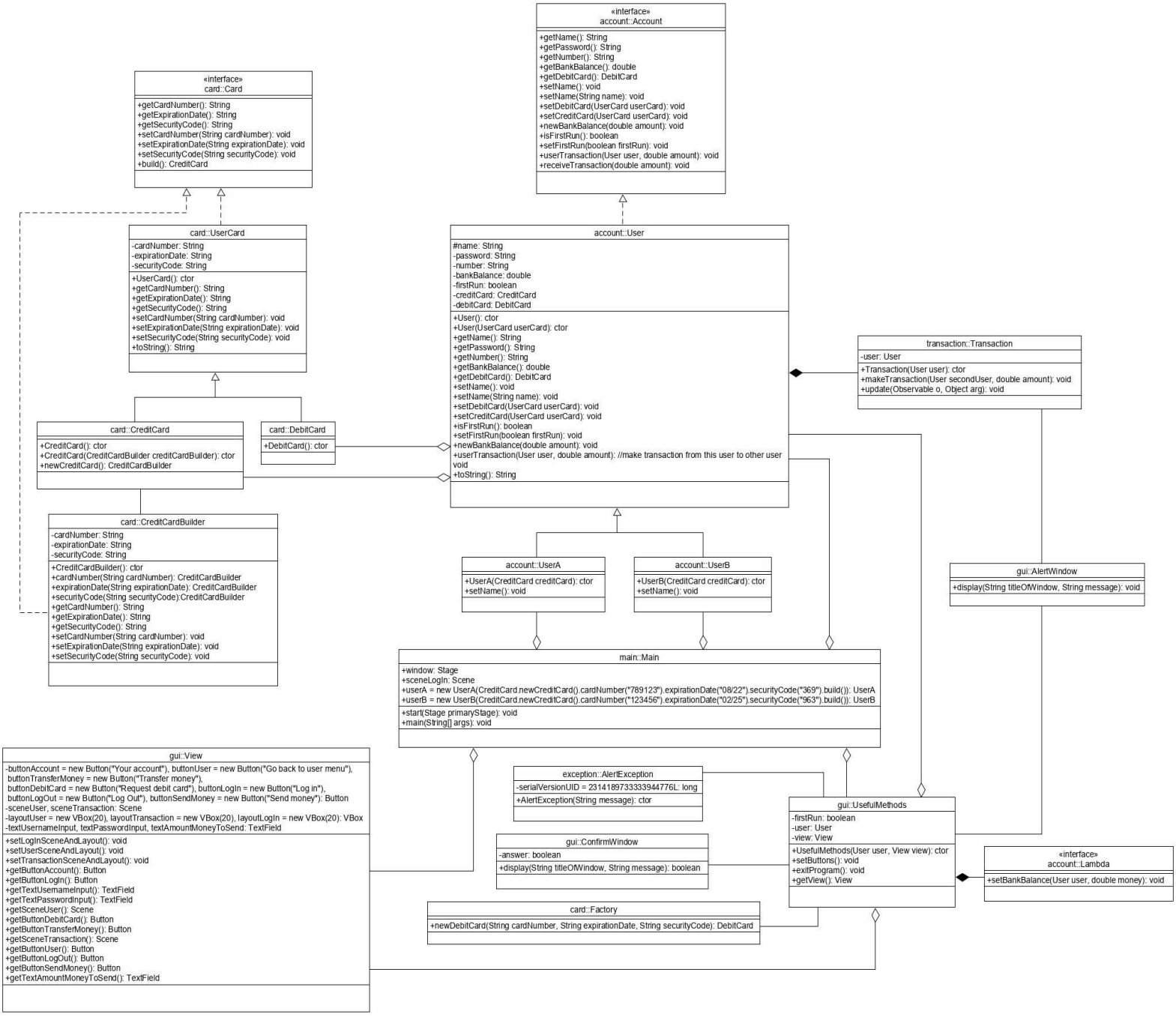
## Zámer projektu – Internetbanking

Projekt Internetbanking rieši simuláciu každodenného kontaktu človeka s bankovým softvérom. Každý užívateľ bude mať dostupný výhradne svoj účet, ktorý sa mu sprístupní po prihlásení. Bezpečnosť bude zabezpečená prihlasovacími údajmi, prihlasovacie meno a heslo, ktoré budú jedinečnými pre každého užívateľa, a tým bude zabránené nahliadnuť do účtu iného používateľa. Jedná sa teda o simuláciu systému na správu citlivých bankových údajov s dôrazom na dôveryhodný softvér aj z hľadiska bezpečnosti. V rámci ukážky budú dostupné dva účty dvoch používateľov.

Užívateľ bude mať dostupné možnosti transakcie, t.j. novej platby, platobných kariet, kde si bude môcť pozrieť typy kariet, aké vlastní – debetná, kreditná – a zostatok na daných platobných kartách(pri debetnej bude zostatok na účte a pri kreditnej povolený limit), ale taktiež aj unikátne údaje na karte(číslo, platnosť a bezpečnostný kód).

V rámci novej platby užívateľ zadá kam posiela peniaze a požadovanú sumu. Pred podpísaním platby sa systém opýta užívateľa, či súhlasí s finančným prevodom. Týmto bude zabezpečený prenos financií medzi jednotlivými užívateľmi. Každý užívateľ bude mať k dispozícii úvodný kapitál, ktorý si bude môcť navýšiť.

# Diagram tried v jazyku UML




Obr. č. 01: Diagram tried

## Vysvetlenie tried a ich vzťahov

- *Main.java* – hlavná trieda cez ktorú je spúšťaný program, obsahuje objekt užívateľa, ale aj objekty pre návrhový vzor mvc(view a controller)
- *User.java* – rodičovská trieda využívajúca rozhranie *Account.java*, definuje správanie 2 používateľov, ktoré je rozšírené v triedach *UserA.java* a *UserB.java*, ktoré dedia práve od *User.java*, agregácia s triedami

*CreditCard.java* a *DebitCard.java* a je tu zároveň využitý návrhový vzor Observer tak, že objekt triedy *User.java* je observable

- *UserCard.java* - rodičovská trieda využívajúca rozhranie *Card.java*, definuje správanie kariet 2 používateľov, samotné karty sú rozšírené v triedach *CreditCard.java* a *DebitCard.java*, ktoré dedia práve od *UserCard.java*
- *Lambda.java* – rozhranie s jednou metódou pre použitie lambda výrazov
- *Factory.java* – implementovaný návrhový vzor Factory
- *Transaction.java* – trieda definujúca samotný prevod peňazí, t.j. transakciu, je tu využitý návrhový vzor Observer, trieda obsahuje objekt užívateľa, ktorý posiela peniaze druhému užívateľovi
- *AlertException.java* – implementácia vlastnej výnimky
- *UsefulMethods.java* – implementovaná kontrola grafického používateľského rozhrania: definovanie správania jednotlivých tlačidiel použitých v grafickom užívateľskom prostredí a zmena zobrazovania cez *View.java*
- *View.java* – nastavenie scenérií grafického používateľského rozhrania pre zobrazenie
- *AlertWindow.java* – implementované upozorňovacie vyskakovacie okno pri nesprávnom užívaní programu(nesprávne zadané prihlasovacie údaje, nesprávne zadaná hodnota novej platby a pod.), trieda je použitá pri vlastných výnimkách
- *ConfirmWindow.java* – implementované potvrdzovacie vyskakovacie okno pri práci s citlivými údajmi(potvrdenie odoslanie platby, potvrdenie ukončenia programu)

## Splnené hlavné kritéria hodnotenia

### Program v súlade so zadaním

Projekt Internetbanking je v súlade so zadaním v nasledujúcich bodoch:

- aplikácia pre prácu s citlivými bankovými údajmi – dôveryhodný softvér
- simulácia prihlasovania 2 používateľov – bezpečný softvér

### Program v súlade so zámerom projektu

Projekt Internetbanking je v súlade so zámerom projektu v nasledujúcich bodoch:

- simulácia práce s bankovým softvérom
- 2 rôzne účty pre 2 rôznych používateľov(užívateľ A, užívateľ B)
- jedinečné prihlasovacie údaje používateľov(prihlasovacie meno a heslo)
- možnosť pozrieť si informácie o účte(meno účtu a zostatok na účte, informácie o kreditnej karte atď.)
- možnosť novej platby - transakcia platby
- potvrdenie platby po podpísaní transakcie
- kreditná i debetná karta s identifikačnými údajmi(číslo karty, jej platnosť a bezpečnostný kód)
- vyžiadanie debitnej karty
- odhlásenie a možnosť prihlásiť sa ako nový používateľ
- prihlásenie iného užívateľa

### Dedenie

Projekt Internetbanking pracuje s objektovo-orientovaným princípom dedenia predovšetkým v dvoch rodičovských triedach *User.java* a *UserCard.java* a ich príslušným triedam, ktoré od nich dedia. Triedy *UserA.java* a *UserB.java* dedia od *User.java* a triedy *CreditCard.java* a *DebitCard.java* dedia od *UserCard.java*.

```java
//subclass UserA inherits from a superclass User
public class UserA extends User{
    public UserA(CreditCard creditCard) {
        super(creditCard);
    }

    @Override
    public void setName() {
        this.name = "UserA";
    }
}
```

```java
//subclass UserB inherits from a superclass User
public class UserB extends User{
    public UserB(CreditCard creditCard) {
        super(creditCard);
    }

    @Override
    public void setName() {
        this.name = "UserB";
    }
}
```

Obr. č. 02: Implementácia dedenia *UserA.java*(vľavo) a *UserB.java*(vpravo) od *User.java*

```java
//subclass CreditCard inherits from a superclass UserCard
public class CreditCard extends UserCard{
    public CreditCard() {
        super();
    }
```

Obr. č. 03: Implementácia dedenia *CreditCard.java* od *UserCard.java*

```java
//subclass DebitCard inherits from a superclass UserCard
public class DebitCard extends UserCard{
    public DebitCard() {
        super();
    }
}
```

Obr. č. 04: Implementácia dedenia *DebitCard.java* od *UserCard.java*

**Polymorfizmus v aspoň 2 oddelených hierarchiách dedenia**

Projekt Internetbanking pracuje s objektovo-orientovaným princípom polymorfizmu v niekoľkých triedach.

Polymorfizmus v rámci hierarchie dedenia *User.java* je napríklad v triede *UsefulMethods.java*, kedy objekt user rodičovskej triedy *User.java* referuje na objekt userB triedy potomka *UserB.java* alebo na objekt userA triedy potomka *UserA.java*. Obdobne je to aj napríklad v triede *Transaction.java*.

```java
//user was choosed as userB
// polymorphism - user object of a superclass User is referencing to a subclass(UserB)
else if(choosedUser.equalsIgnoreCase("b")) this.user = Main.userB;

//user was choosed as userA
// polymorphism - user object of a superclass User is referencing to a subclass(UserA)
else if(choosedUser.equalsIgnoreCase("a")) this.user = Main.userA;
```

Obr. č. 05: Implementácia polymorfizmu v rámci hierarchie dedenia rodičovskej triedy *User.java* v triede *UsefulMethods.java*

```java
//make transaction from this user to secondUser
//polymorphism - secondUser object of a superclass User is referencing to a subclass(UserA or UserB)
public void makeTransaction(User secondUser, double amount) {
    //exception - amount of money to send is either too big or too small
    //exception is caught in User.java
    if(amount > this.user.getBankBalance() || amount <= 0) throw new AlertException("You can not send this amount of money!");

    this.user.newBankBalance(-amount);
    secondUser.receiveTransaction(amount);  //polymorphism - secondUser object of a superclass User is referencing to a subclass(UserA or UserB)
}
```

Obr. č. 06: Implementácia polymorfizmu v rámci hierarchie dedenia rodičovskej triedy *User.java* v triede *Transaction.java*

Polymorfizmus v rámci hierarchie dedenia *UserCard.java* je napríklad v triede *User.java*, kedy objekt userCard rodičovskej triedy *UserCard.java* referuje na objekt triedy potomka *CreditCard.java*, neskôr nastáva pretypovanie na príslušnú triedu, tzv. downcasting v zapuzdrenej metóde typu set pre prístup k objektu creditCard.

```java
//constructor overloading to create 2 static final objects in Main.java
//userCard is polymorphism - object userCard of a superclass references to a object of a subclass CreditCard or DebitCard
public User(UserCard userCard) {
    this.setName();
    this.setCreditCard(userCard);   //polymorphism - object userCard of a superclass references to a object of a subclass CreditCard
    this.setDebitCard(null);
    this.setFirstRun(true);
}
```

Obr. č. 07: Implementácia polymorfizmu v rámci hierarchie dedenia rodičovskej triedy *UserCard.java* v triede *User.java*

### Rozhrania

Projekt Internetbanking implementuje objektovo-orientovaný princíp rozhrania v troch rozhraniach: *Account.java*, *Card.java* a *Lambda.java*, pričom *Lambda.java* je funkcionálne rozhranie(rozhranie pracujúce s jednou abstraktnou metódou, ktorá je využitá pri lambda výrazoch).

### Zapuzdrenie

Projekt Internetbanking pracuje s objektovo-orientovaným princípom zapuzdrenia v niekoľkých triedach.

```java
//encapsulation
//get method
public Button getButtonAccount() {
    return buttonAccount;
}

//encapsulation
//get method
public Button getButtonLogIn() {
    return buttonLogIn;
}

//encapsulation
//get method
public TextField getTextUsernameInput() {
    return textUsernameInput;
}
```

```java
//encapsulation
//get method
public TextField getTextPasswordInput() {
    return textPasswordInput;
}

//encapsulation
//get method
public Scene getSceneUser() {
    return sceneUser;
}

//encapsulation
//get method
public Button getButtonDebitCard() {
    return buttonDebitCard;
}
```

Obr. č. 08: Implementácia niektorých zapuzdrení v triede *View.java*

### Agregácia

Projekt Internetbanking pracuje s objektovo-orientovaným princípom agregácie v niekoľkých triedach, napríklad v rodičovskej triede *User.java* sa nachádzajú objekty creditCard triedy *CreditCard.java* a debitCard triedy *DebitCard.java*.

```java
// aggregation
private CreditCard creditCard;
```

```java
// aggregation
private DebitCard debitCard;
```

Obr. č. 09: Implementácia agregácie v rodičovskej triede *User.java*

### Oddelenie aplikačnej logiky od používateľského rozhrania

Projekt Internetbanking pracuje s oddelením aplikačnej logiky od používateľského prostredia. Grafické používateľské prostredie je sprostredkované prostredníctvom *javafx* v samostatnom balíku gui a príslušných triedach:

- *AlertWindow.java* – vyskakovacie upozorňovacie okno,
- *ConfirmWindow.java* – vyskakovacie potvrdzujúce okno,
- *UsefulMethods.java* – controller z návrhového vzoru mvc, aplikačná logika,
- *View.java* – view z návrhového vzoru mvc, nastavenie samotného grafického používateľského rozhrania zobrazeného používateľovi.

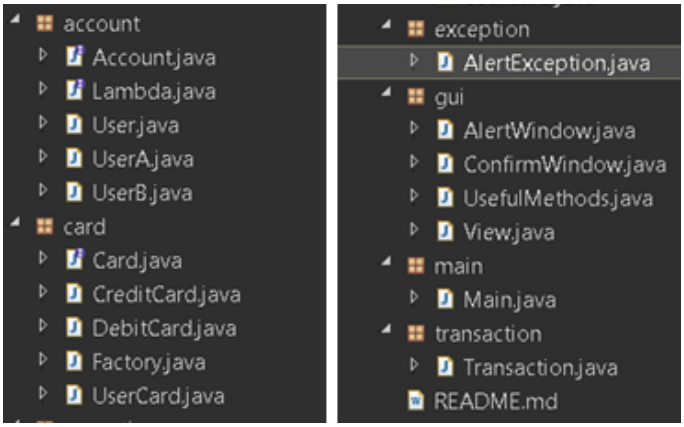
V triede *Main.java* je definované spustenie programu.

```
//separate gui from logic
View view = new View();                                    //object view as view in mvc pattern
User user = new User();                                    //object user as model in mvc pattern
UsefulMethods controller = new UsefulMethods(user, view);  //object controller as controller in mvc pattern
```

Obr. č. 10: Implementácia návrhového vzoru mvc v triede *Main.java*

### Kód vhodne organizovaný do balíkov

Projekt Internetbanking pracuje s kódom vhodne organizovaným do rôznych balíkov.




Obr. č. 11: Organizácia tried v balíkoch

### Prehľadná dokumentácia so všetkými položkami podľa opisu

Projekt Internetbanking v rámci tejto dokumentácie obsahuje tieto položky:

- meno a priezvisko autora
- názov a zámer projektu
- štruktúra systému vo forme diagramu najdôležitejších tried s vysvetlením tried a ich vzťahov
- plnenie kritérií hodnotenia – hlavné i ďalšie

- zoznam hlavných verzií programu odovzdaných do GitHub classroom s krátkym opisom najdôležitejších zmien pre každú verziu
- Javadoc dokumentácia

Všetky tieto položky sú definované na príslušnej internetovej stránke predmetu a všetky tieto položky dokumentácia obsahuje.

## Splnené ďalšie kritéria hodnotenia

### Návrhové vzory(Observer, Builder a Factory)

V podmienkach hodnotenia je možnosť použitia návrhových vzorov okrem návrhového vzoru Singleton.

**Observer**

Projekt Internetbanking pracuje s objektovo-orientovaným princípom návrhového vzoru Observer, ktorý špecifikuje komunikáciu medzi objektmi observable a observer. Objekt observable je objekt, ktorý upozorňuje observer ohľadom zmeny ich stavu.

Rodičovská trieda *User.java* funguje ako observable, t.j. ak je zmenený finančný stav(transakcia) jedného z užívateľov, tak je upozornený observer, v tomto prípade objekt triedy *Transaction.java*.

```
@Override
//make transaction from this user to other user
public void userTransaction(User user, double amount) {
    Transaction transaction = new Transaction(this);

    //exception - if there is an exception in Transaction class it will be thrown from makeTransaction method in Transaction class
    try {
        transaction.makeTransaction(user, amount);
        this.addObserver(transaction);  //observer - transaction object
        setChanged();
        notifyObservers(user);          //notify transaction object with argument other user(he will get money from this user, an observable object)
        deleteObserver(transaction);
    }

    //exception caught
    catch(AlertException alert) {
        AlertWindow.display("ALERT", alert.getMessage());
    }
}
```

Obr. č. 12: Implementácia metódy objektu observable v triede *User.java*

```
@Override
public void update(Observable o, Object arg) {
    User secondUser = (User) arg;   // polymorphism - secondUser object of a superclass User is referencing to a subclass(UserA or UserB)

    AlertWindow.display("Transaction", "Transaction successful. " + this.user.getName() +
            " sent money and " + secondUser.getName() + " received money.");
}
```

Obr. č. 13: Implementácia metódy objektu observer v triede *Transaction.java*

**Builder**

Projekt Internetbanking pracuje s objektovo-orientovaným princípom návrhového vzoru Builder, ktorý poskytuje vytvorenie objektu triedy *CreditCard.java*. Samotné vytvorenie sa nachádza v triede *Main.java* a implementácia sa nachádza v triede *CreditCard.java* ako vhniezdená trieda.

```
//User´s credit card created with design pattern Builder
public static final UserA userA = new UserA(CreditCard.newCreditCard()
        .cardNumber("789123")
        .expirationDate("08/22")
        .securityCode("369")
        .build());
```

Obr. č. 14: Vytvorenie objektu userA triedy *UserA.java* s vytvorením nového objektu kreditnej karty triedy *CreditCard.java* pomocou návrhového vzoru Builder

```
//User´s credit card created with design pattern Builder
public static final UserB userB = new UserB(CreditCard.newCreditCard()
        .cardNumber("123456")
        .expirationDate("02/25")
        .securityCode("963")
        .build());
```

Obr. č. 15: Vytvorenie objektu userB triedy *UserB.java* s vytvorením nového objektu kreditnej karty triedy *CreditCard.java* pomocou návrhového vzoru Builder

```
//design pattern Builder
//nested class
public static class CreditCardBuilder implements Card{
    private String cardNumber;
    private String expirationDate;
    private String securityCode;

    public  CreditCardBuilder() {
    }

    public CreditCardBuilder cardNumber(String cardNumber) {
        this.setCardNumber(cardNumber);
        return this;
    }

    public CreditCardBuilder expirationDate(String expirationDate) {
        this.setExpirationDate(expirationDate);
        return this;
    }

    public CreditCardBuilder securityCode(String securityCode) {
        this.setSecurityCode(securityCode);
        return this;
    }
```

Obr. č. 16: Implementácia návrhového vzoru Builder v triede *CreditCard.java* je s použitím vhniezdenej triedy

**Factory**

Projekt Internetbanking pracuje s objektovo-orientovaným princípom návrhového vzoru Factory, ktorý poskytuje vytvorenie objektu triedy *DebitCard.java*.

```
//debit card has not been selected yet for userA
if(this.user.getName().equals("UserA")) {
    if(this.user.getDebitCard() == null) {
        AlertWindow.display("REQUEST DEBIT CARD", "Your debit card was added to your account, " + this.user.getName() + "!");
        this.user.setDebitCard(Factory.newDebitCard("1111111", "01/30", "111"));    //Factory pattern to create debit card
        lambda.setBankBalance(this.user, 1000);
        getView().getButtonDebitCard().setText("You already have a debit card");
    }
}

//debit card has not been selected yet for userB
if(this.user.getName().equals("UserB")) {
    if(this.user.getDebitCard() == null) {
        AlertWindow.display("REQUEST DEBIT CARD", "Your debit card was added to your account, " + this.user.getName() + "!");
        this.user.setDebitCard(Factory.newDebitCard("222222", "02/30", "222"));     //Factory pattern to create debit card
        lambda.setBankBalance(this.user, 1000);
        getView().getButtonDebitCard().setText("You already have a debit card");
    }
}
```

Obr. č. 17: Vytvorenie debetnej karty pre objekty userA a userB v triede *UsefulMethods.java* s pomocou návrhového vzoru Factory

```
//design pattern Factory
public class Factory {
    public static DebitCard newDebitCard(String cardNumber, String expirationDate, String securityCode) {
        DebitCard DebitCard = new DebitCard();
        DebitCard.setCardNumber(cardNumber);
        DebitCard.setExpirationDate(expirationDate);
        DebitCard.setSecurityCode(securityCode);

        return DebitCard;
    }
}
```

Obr. č. 18: Implementácia návrhového vzoru Factory v triede *Factory.java*

### Ošetrenie mimoriadnych stavov prostredníctvom vlastných výnimiek

Projekt Internetbanking pracuje s objektovo-orientovaným princípom vlastných výnimiek v niekoľkých triedach, *Transaction.java*, *UsefulMethods.java*, *User.java*, a vlastná výnimka je definovaná v triede *AlertException.java*. Vyhadzovanie vlastnej výnimky je možné vidieť na obr. č. 06, obr. č. 12 a na obr. č. 20.

```
public class AlertException extends RuntimeException{
    private static final long serialVersionUID = 2314189733333944776L;

    public AlertException(String message) {
        super(message);
    }
}
```

Obr. č. 19: Implementácia vlastnej výnimky

```java
try {
    //exception - the amount of money has not been written in the text field
    if(checkNumber.isEmpty()) throw new AlertException("Text field is empty! Please try again.");

    else {
        //cycle to find if there is , or a letter
        for (int i = 0; i < checkNumber.length(); i++) {

            //exception - there is a letter
            if( Character.isLetter( checkNumber.charAt(i) ) ) throw new AlertException(String.format("%s is not number! Please try again.", checkNumber));

            //exception - there is ,
            else if( ',' == checkNumber.charAt(i) )
                throw new AlertException(String.format("You can not use , in %s! Please try again.", checkNumber));
        }

        //text in the text field is a valid number
        //transfer String object to Double object - wrapper class
        Double amount = new Double(checkNumber);

        //send money to the userB from userA
        if("UserA".equalsIgnoreCase(this.user.getName())) this.user.userTransaction(Main.userB, amount);

        //send money to the userA from userB
        else this.user.userTransaction(Main.userA, amount);
    }
}

//exception has been caught
catch(AlertException alert) {
    AlertWindow.display("ALERT", alert.getMessage());
}
```

Obr. č. 20: Vyhadzovanie vlastnej výnimky v triede *UsefulMethods.java* pri odosielaní peňazí – transakcii

### Použitie implicitnej implementácie metód v rozhraniach

Projekt Internetbanking pracuje s objektovo-orientovaným princípom implicitnej implementácie metód v rozhraniach, tzv. default method implementation. V oboch rozhraniach(*Account.java* a *Card.java*) je použitá implicitná implementácia metód.

```java
//default method for user to receive transaction
public default void receiveTransaction(double amount) {
    this.newBankBalance(amount);
}
```

Obr. č. 21: Default method implementation v rozhraní *Account.java*

```java
//default build method for design pattern Builder for CreditCardBuilder
public default CreditCard build() {
    return new CreditCard((CreditCardBuilder) this);
}
```

Obr. č. 22: Default method implementation v rozhraní *Card.java*

### Poskytnutie grafického používateľského rozhrania oddelene od aplikačnej logiky a s aspoň časťou spracovateľov udalostí (handlers) vytvorenou manuálne

Projekt Internetbanking pracuje s objektovo-orientovaným princípom spracovateľa udalostí, tzv. event hadler v triede *UsefulMethods.java*, t.j. mechanizmus, ktorý kontroluje udalosť a rozhoduje, čo sa má stať, ak udalosť nastane.

```
//go to the menu for transferring the money
getView().getButtonTransferMoney().setOnAction(e -> Main.window.setScene(getView().getSceneTransaction()));

//go back to the user´s menu
getView().getButtonUser().setOnAction(e -> Main.window.setScene(getView().getSceneUser()));

//log out and go back to the main menu
getView().getButtonLogOut().setOnAction(e->{
    AlertWindow.display("LOG OUT", "You successfully logged out of your bank account, " + this.user.getName() + "!");
    this.user = null;
    Main.window.setScene(Main.sceneLogIn);
});
```

Obr. č. 23: Príklady implementácií event handlers v triede *UsefulMethods.java*

Projekt Internetbanking pracuje s oddelením aplikačnej logiky od používateľského prostredia. Grafické používateľské prostredie je sprostredkované prostredníctvom *javafx* v samostatnom balíku gui a príslušných triedach:

- *AlertWindow.java* – vyskakovacie upozorňovacie okno,
- *ConfirmWindow.java* – vyskakovacie potvrdzujúce okno,
- *UsefulMethods.java* – controller z návrhového vzoru mvc, aplikačná logika,
- *View.java* – view z návrhového vzoru mvc, nastavenie samotného grafického používateľského rozhrania zobrazeného používateľovi.

V triede *Main.java* je definované spustenie programu.

```
//separate gui from logic
View view = new View();                                  //object view as view in mvc pattern
User user = new User();                                  //object user as model in mvc pattern
UsefulMethods controller = new UsefulMethods(user, view);   //object controller as controller in mvc pattern
```

Obr. č. 24: Implementácia návrhového vzoru mvc v triede *Main.java*

### Použitie vhniezdených tried a rozhraní

Projekt Internetbanking pracuje s objektovo-orientovaným princípom použitia vhniezdených tried v triede *CreditCard.java*, kde je vytvorená statická trieda *CreditCardBuilder*.

```java
//subclass CreditCard inherits from a superclass UserCard
public class CreditCard extends UserCard{
    public CreditCard() {
        super();
    }

    //Builder pattern
    public CreditCard(CreditCardBuilder creditCardBuilder) {
        setCardNumber(creditCardBuilder.getCardNumber());
        setExpirationDate(creditCardBuilder.getExpirationDate());
        setSecurityCode(creditCardBuilder.getSecurityCode());
    }

    //Builder pattern
    public static CreditCardBuilder newCreditCard() {
        return new CreditCardBuilder();
    }

    //design pattern Builder
    //nested class
    public static class CreditCardBuilder implements Card{
        private String cardNumber;
        private String expirationDate;
        private String securityCode;

        public  CreditCardBuilder() {
```

Obr. č. 25: V triede *CreditCard.java* je použitá vhniezdená trieda CreditCardBuilder

### Lambda výrazy

Projekt Internetbanking pracuje s objektovo-orientovaným princípom použitia lambda výrazov v triede *UsefulMethods.java* a použité je pritom rozhranie *Lambda.java* s jednou abstraktnou metódou, čo umožňuje použitie lambda výrazov.

```java
//request debit card for user
getView().getButtonDebitCard().setOnAction(e -> {
    Lambda lambda = (User user, double money) -> user.newBankBalance(money);    //lambda expression

    //debit card has not been selected yet for userA
    if(this.user.getName().equals("UserA")) {
        if(this.user.getDebitCard() == null) {
            AlertWindow.display("REQUEST DEBIT CARD", "Your debit card was added to your account, " + this.user.getName() + "!");
            this.user.setDebitCard(Factory.newDebitCard("1111111", "01/30", "111"));    //Factory pattern to create debit card
            lambda.setBankBalance(this.user, 1000);     //lambda expression to increase bank account when debit card created
            getView().getButtonDebitCard().setText("You already have a debit card");
        }
    }

    //debit card has not been selected yet for userB
    if(this.user.getName().equals("UserB")) {
        if(this.user.getDebitCard() == null) {
            AlertWindow.display("REQUEST DEBIT CARD", "Your debit card was added to your account, " + this.user.getName() + "!");
            this.user.setDebitCard(Factory.newDebitCard("222222", "02/30", "222"));     //Factory pattern to create debit card
            lambda.setBankBalance(this.user, 1000);     //lambda expression to increase bank account when debit card created
            getView().getButtonDebitCard().setText("You already have a debit card");
        }
    }
});
```

Obr. č. 26: Použitie lambda výrazov v triede *UsefulMethods.java*

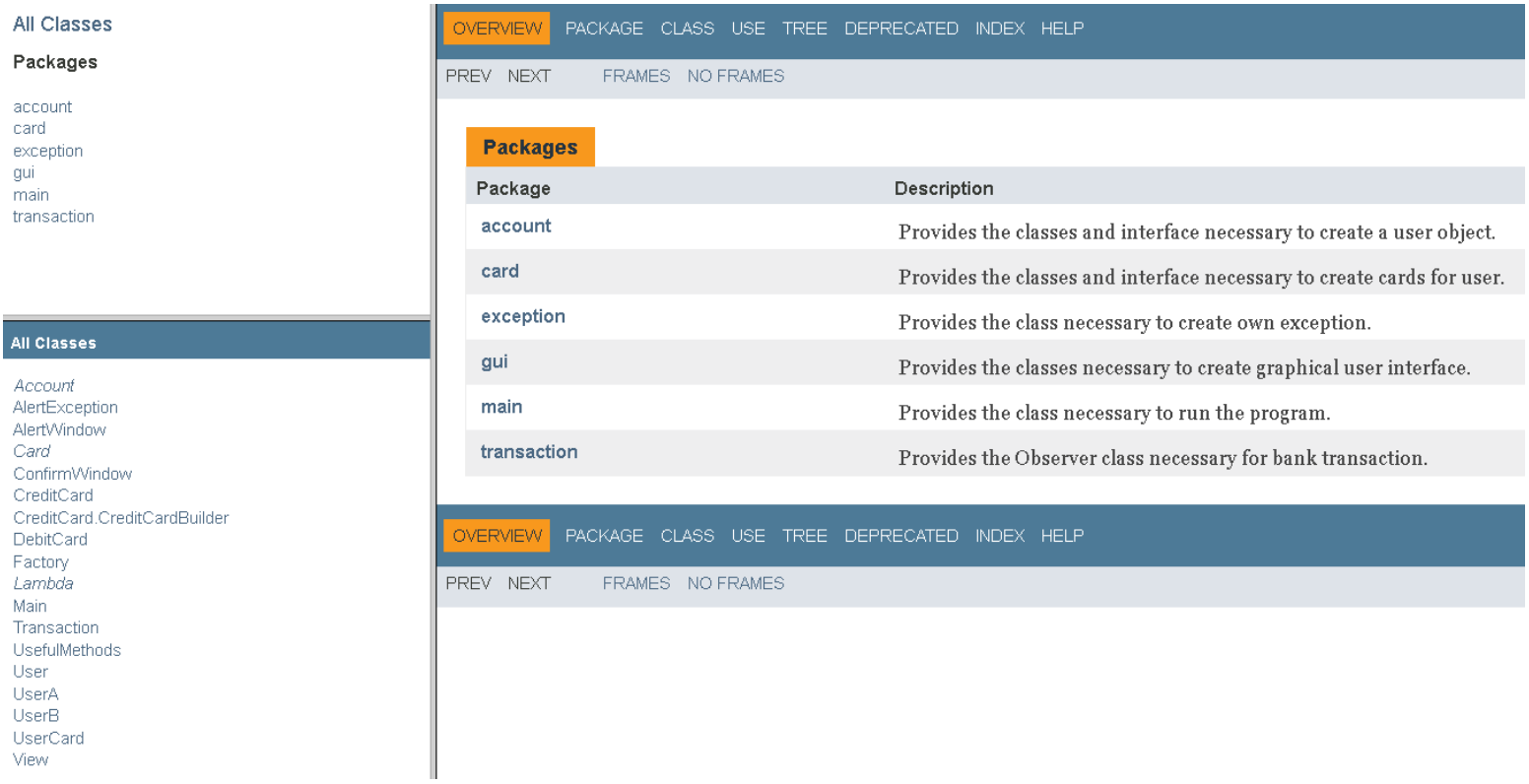
## Zoznam hlavných verzií programu odovzdaných do GitHub classroomu

Zoznam odovzdaných verzií do GitHub classroom do vetvy master, zoradené od prvého komitu:

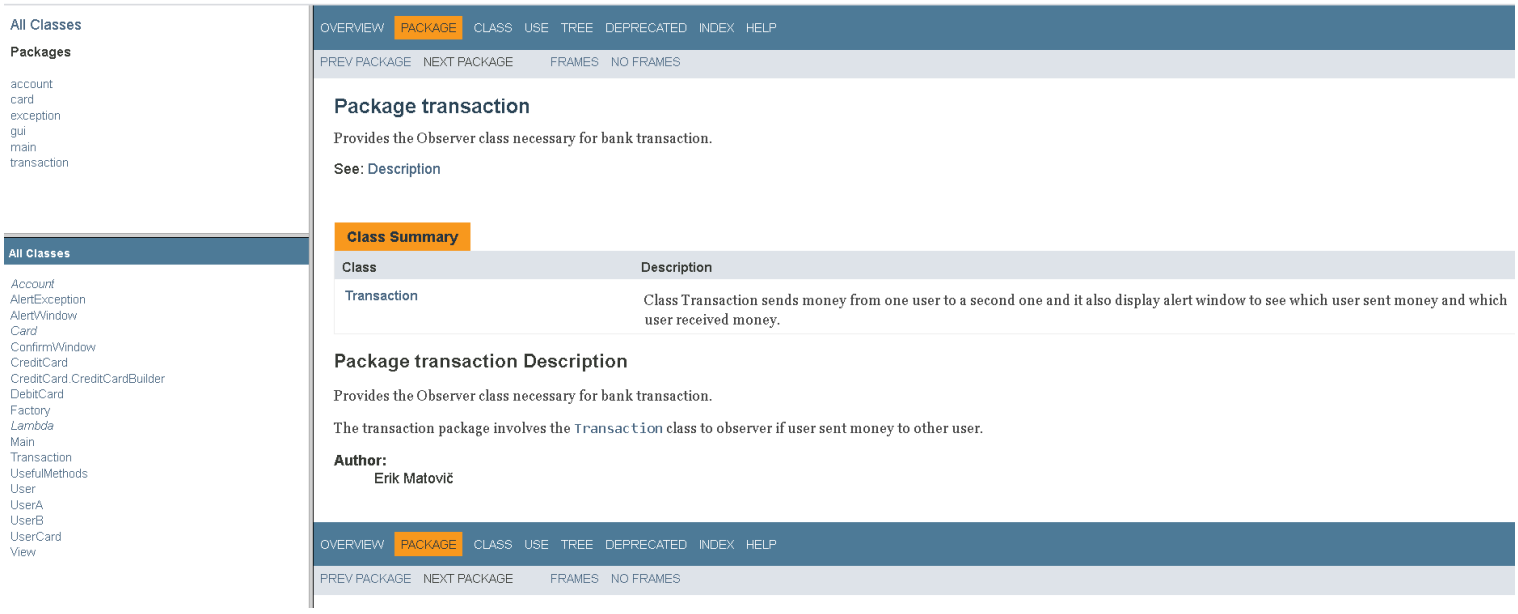
- **first version** – 24.03.2020 – prvá verzia programu, obsahujúca rozhrania *Account.java* a *Card.java*, triedu *Main.java*, dedenie od rodičovskej triedy *User.java*, ovládanie cez konzolu
- **card implemented to the project** – 31.03. 2020 – implementovaná funkcia kreditnej karty, pridané zapuzdrenie a agregácia, ovládanie stále cez konzolu
- **observer pattern added** – 06.04.2020 – implementovaná trieda *Transaction.java* a návrhový vzor Observer
- **gui added** – 14.04.2020 – zmena užívateľské prostredia z ovládania cez konzolu na grafické používateľské rozhranie implementované cez *javafx*
- **Factory pattern and Builder pattern added** – 22.04.2020 – implementované návrhové vzory Factory a Builder
- **exception added** – 30.04.2020 – pridané vlastné výnimky, implementovaná trieda *AlertException.java* cez balík exception
- **UserCard.java added** – 14.05.2020 – implementovaná rodičovská trieda *UserCard.java* a reorganizácia tried *CreditCard.java* a *DebitCard.java* tak, aby dedili od *UserCard.java*, úprava pre splnenie hlavného kritéria hodnotenia polymorfizmu v 2 rôznych hierarchiách dedenia
- **default method implementation added** – 17.05.2020 – zakomponované implicitné implementácie metód v rozhraniach
- **lambda added** – 18.05.2020 – implementované lambda výrazy
- **code refactoring** – 19.05.2020 – refactoring kódu, pridanie komentárov, doriešenie nechcených stavov

## Javadoc dokumentácia

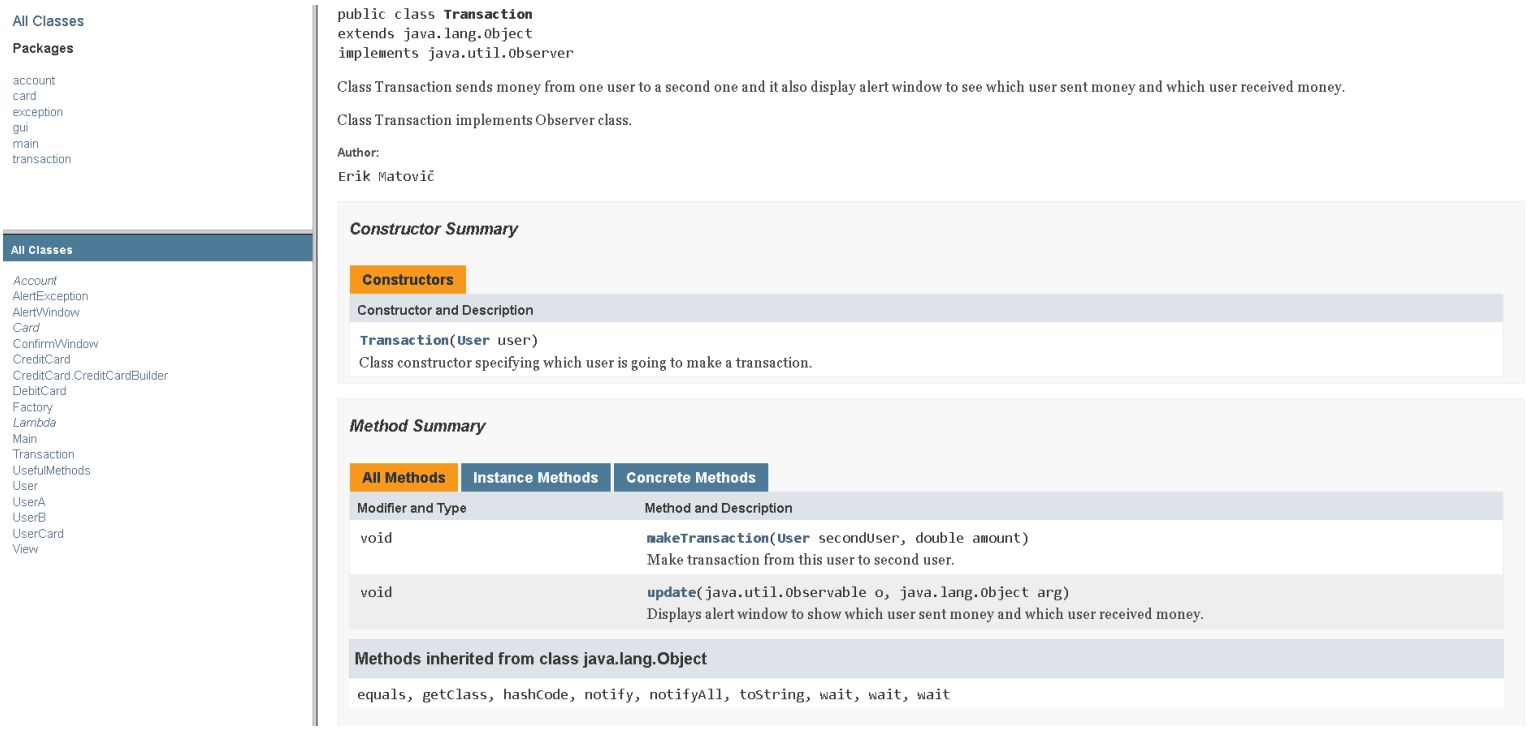
Javadoc dokumentácia je dokumentácia generovaná priamo z programu nástrojom javadoc vo formáte HTML. Celá dokumentácia je dostupná v classroom na GitHube a spustiteľná je cez *index.html*.




Obr. č. 27: Javadoc overview




Obr. č. 28: Javadoc package transaction



All Classes

Packages

account
card
exception
gui
main
transaction

All Classes

Account
AlertException
AlertWindow
Card
ConfirmWindow
CreditCard
CreditCard.CreditCardBuilder
DebitCard
Factory
Lambda
Main
Transaction
UsefulMethods
User
UserA
UserB
UserCard
View

public class **Transaction**
extends java.lang.Object
implements java.util.Observer

Class Transaction sends money from one user to a second one and it also display alert window to see which user sent money and which user received money.

Class Transaction implements Observer class.

**Author:**
Erik Matovič

***Constructor Summary***

**Constructors**

| Constructor and Description |
| --- |
| **Transaction**(**User** user)<br>Class constructor specifying which user is going to make a transaction. |

***Method Summary***

**All Methods** | **Instance Methods** | **Concrete Methods**

| Modifier and Type | Method and Description |
| --- | --- |
| void | **makeTransaction**(**User** secondUser, double amount)<br>Make transaction from this user to second user. |
| void | **update**(java.util.Observable o, java.lang.Object arg)<br>Displays alert window to show which user sent money and which user received money. |

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Obr. č. 29: Javadoc Class Transaction

```java
/**
* Class Transaction sends money from one user
* to a second one and it also display alert window
* to see which user sent money and which user received money.
* <p>
* Class Transaction implements Observer class.
* @author Erik Matovič
*/
public class Transaction implements Observer{
    private User user;

    /**
    * Class constructor specifying which user
    * is going to make a transaction.
    * <p>
    * @param  user  this user is trying to make a transaction
    * @author Erik Matovič
    */
    public Transaction(User user) {
        this.user = user;
    }

    /**
    * Make transaction from this user to second user.
    * <p>
    * This method contains polymorphism - secondUser object
    * of a superclass User is referencing to a subclass(UserA or UserB)
    * @param  secondUser    a user to receive money
    * @param  amount        amount of money to be send
    * @see                  User
    * @author Erik Matovič
    */
    public void makeTransaction(User secondUser, double amount) {
```

Obr. č. 30: Komentáre potrebné pre vygenerovanie dokumentácie Javadoc v triede *Transaction.java*