TEHNICI DE PROGRAMARE

Simularea unei banci

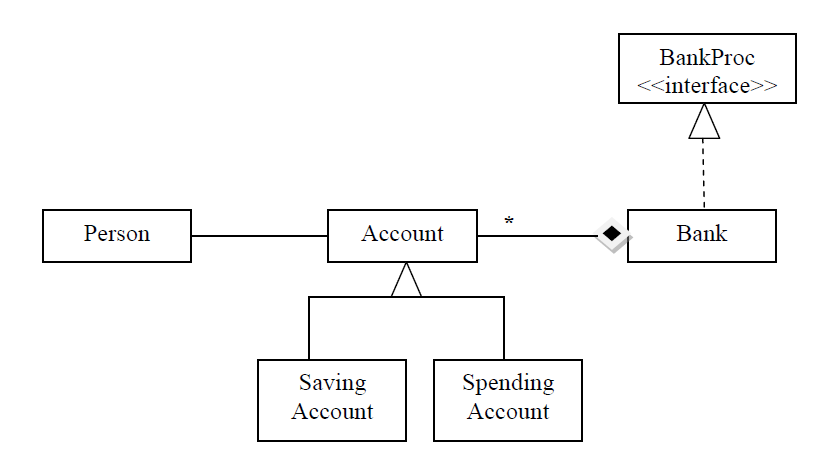
Gozar-Manu Ariana

An II , CTI - romana

Grupa 30225

**Cerinta**

Realizati proiectul dupa urmatoarea schema :



* Implementati urmatoarele metode : adaugare / dezactivare / reactivare cont , extragere / depunere de numerar , interogare sold , cautare dupa diferite campuri etc .
* Implementati Clasa Bank cu ajutorul unei tabele de dispersie .
* Populati banca dintr-un fisier.
* Folositi pre si post conditii – Design by contract

**1. Obiectivul temei**

Ne propunem sa implementam un sistem software bazat pe ideea unei banci in care se pot adauga conturi, se pot sterge si se pot efectua diferite operatiuni ca si depunerea, extragerea de numerar pe aceste conturi , interogare sold si alte diferite operatii aditionale .

Acest sistem va fi asemanatori unui ATM (prescurtare a sintagmei engleze **Automated Teller Machine** – literal *„casier automat”*) sau bancomat , adica , un aparat electronic sau electromecanic care permite clienților să efectueze diferite tranzacții bancare prin autoservire .

**2. Analiza problemei, modelare, scenarii, cazuri de utilizare**

Pentru a rezolva problema am implementat 13 clase : Person , Account , Bank , SavingAccount , SpendingAccount , Main , BankProc , FirstFrame , WriteFile , Read File , CreateAccFrame , SpendingAccFrame , SavingAccFrame .

In clasa Person am modelat un client , care pastreaza informatii specifice pentru fiecare client cum ar fi nume , adresa si CNP .

In clasa Account am modelat un cont , avand urmatoarele campuri : PIN – codul pin al contului prin interdiul caruia un client isi poate accesa contul si prin care se identifica campul unic din HashTable ; client – ca fiind un obiect de tip Person si anume titularul contului si tipul contului care poate fi de doua feluri : Saving Account si Spending Account .

Clasele Saving Account si Spending Account extind clasa Account , adica o mostenesc si contin metoda de depunere numerar , respectiv de extragere numerar si de obtinere a sumelor de bani aflate in conturi .

Clasa Bank reprezinta banca . Pentru memorarea conturilor am folosit o tabela de dispersie , care are ca si cheie codul pin al contului si un camp care contine fiecare cont corespunzatori acelui cod pin . Pentru o buna accesare a datelor am retinut si intr-un ArrayList datele pentru fiecare cont .

Clasa Main se ocupa cu activarea primului frame folosit pe care il vede utilizatorul si anume FirstFrame . De asemenea , aici se va apela metoda de populare din clasa ReadFile , prin care se va popula HashTable-ul cu elemente din fisier .

Clasa BankProc este o interfata in care sunt definite cateva metode pe care Clasa Bank este neaparat sa le implementeze : createSavingAccount – in care se creaza un cont de tip Saving Account ; createSpendingAccount – in care se creaza un cont de tip Spending Account ; deactivateAccount – pentru a dezactiva un cont ; activateAccount – prin intermediul careia se activeaza un cont daca acesta a fost dezactivat , astfel fiecare cont nu se pierde , ci se inchide pentru o vreme , proprietarul putandu-l refolosi atunci cand doreste ; extractMoney – care e de fapt o metoda de legatura pentru a accesa conturile de tip Saving Account si Spending Account si depositMoney care este la fel , doar ca prima se ocupa de extragerea de bani , adica face legatura cu metodele de tip getMoney , iar cea de a doua face legatura cu metodele de tip putMoney .

Clasa FirstFrame implementeaza interfata grafica pe care o vede utilizatorul prima data la rularea aplicatiei . Aceasta permite sa-si aleaga clientul ce cont doreste sa acceseze , sa-si creeze un cont nou sau sa activeze / dezactiveze vreun cont in functie de pin-ul acelui cont .

Clasa CreateAccFrame implementeaza un frame care apare atunci cand se acceseaza butonul create din interfata implementata in clasa FirstFrame . In acest frame vor aparea cateva campuri obligatorii care trebuie completate pentru a putea crea un cont . Dupa ce se creaza contul va aparea un mesaj cu pin-ul acelui cont , iar pentru a parasi acest frame se apasa butonul EXIT .

Clasa SpendingAccFrame este accesata atunci cand se da click pe butonul Spending Account din interfata implementata in clasa FirstFrame . Frame-ul aparut va contine trei butoane prin care clientul isi poate alege ce doreste sa faca si pe margine apar cateva JRadioButton-uri prin care poate sa-si aleaga ce suma sa extraga sau sa depuna din / in cont . De asemenea , acest cont , are ceva special , deoarece , daca clientul doreste sa-si impuna o anumita limita pentru acest cont , el o poate face , iar banii depusi pe langa se vor transfera automat in cont-ul sau de Saving introducand doar pin-ul acelui cont . La fel , pentru a iesi din acest frame trebuie accesat butonul de inchidere .

Clasa SavingAccFrame este asemanatoare celei descrise anterior , atat doar ca aceasta nu are acea limita pe care o poate impune clientul . De asemenea , diferenta majora dintre aceste doua conturi o face faptul ca la Saving Account primesti dobanda atunci cand introduci bani in el , iar la Spending Account ti se extrage comision de fiecare data cand scoti bani din banca .

Clasa ReadFile are doar o singura metoda , si anume , populate , cu ajutorul careia se face popularea HashTable-ului cu date dintr-un fisier .

Clasa WriteFile are si ea , la randul sau , doar o metoda , prin care se salveaza toate datele aflate in HashTable in fisier .

Pentru implementare s-a folosit si metoda Design by Contract care foloseste pre si post conditii si asertiuni - expresii ce reprezintă condiţii considerate ca trebuind să fie adevărate în anumite puncte de program :

• Dacă o astfel de condiţie nu e satisfăcută, se semnalează eroare la execuţie ;

• Două forme sintactice pentru instrucţiunea assert :

assert expression1 ;

assert expression1 : expression2 ;

* unde expression1 : expresie cu valoare booleană ;
* iar expression2 : orice expresie care are valoare ( nu poate fi apel de metodăcare returnează void ) ;

• Dacă la execuţia lui assert valoarea expression1 este true , execuţia continuă normal , altfel se generează excepţia neverificabilă AssertionError ;

Excepţia AssertionError poate fi interceptată în program ; dacă nu , se va afişa un mesaj de forma : Exception in thread “main” java.lang.AssertionError at ProgramName.main(ProgramName.java :23) ;

• Dacă este prezentă expression2 , reprezentarea ei ca String se adaugă după AssertionError în mesajul de eroare ( explică mai complet natura erorii ) ;

• Aserţiunile permit programare defensivă : reducerea efectului erorilor chiar când nu e cunoscută localizarea acestora ;

• Se folosesc în mod normal numai pentru faza de testare a unei aplicaţii nu trebuie să aibă efecte secundare ;

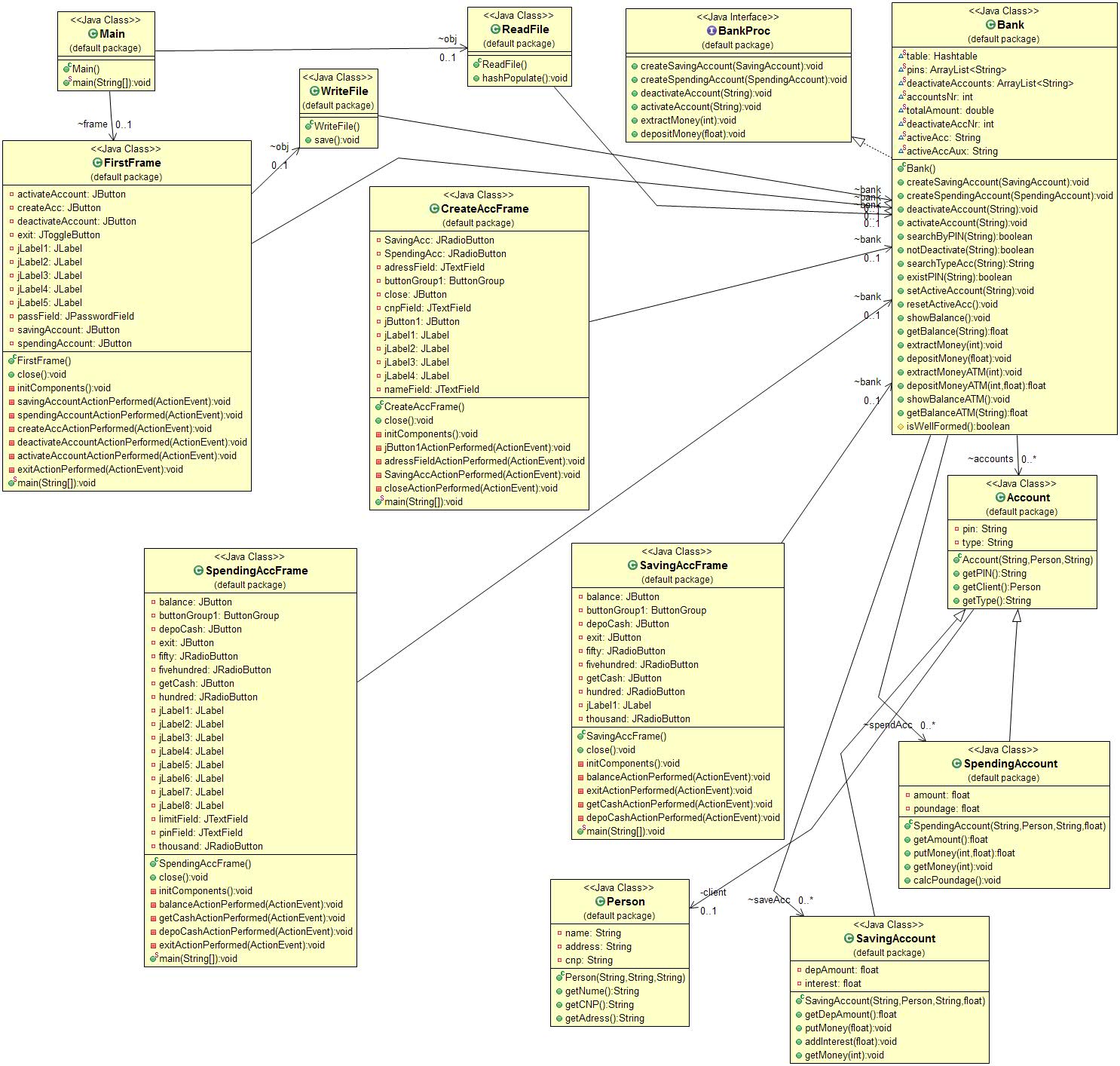
• Trei forme principale de utilizare: – La începutul unei metode , pentru a verifica dacă e apelată conform contractului ( precondiţie ) – La sfârşitul unei metode , pentru a verifica dacă rezultatul e plauzibil ( postcondiţie ) – În corpul unei metode , pentru a verifica dacă anumite proprietăţi sunt afectate ( invarianţi ) ;

**3. Proiectare**

Clasele folosite le-am descris mai sus , in sectiunea anterioara . Pentru fiecare clasa voi da mai multe detalii despre metodele lor acuma :

* SpendingAccount are 4 metode : getAmount este folosita pentru a returna cantitatea numerica care se afla in acel moment , al accesarii interogarii soldului , in contul accesat de client ; putMoney este o metoda comuna tuturor tipurilor de conturi si se ocupa cu adaugarea unei valori de tip int sumei actuale aflate in cont . Aceasta are si un parametru de tip float care este , de fapt , limita impusa de client pentru constul de Spending , restul de bani depusi transferandu-se in contul specificat de client de Saving . Astfel , in aceasta metoda se calculeaza si diferenta dintre limita si banii depusi , returnand aceasta diferenta pentru a putea fi transferata mai departe ; getMoney este , de asemenea , tot o metoda comuna tuturor tipurilor de conturi , ocupandu-se cu extragerea unei anumite sume transmise ca parametru , de tip int ; calcPoundage este metoda accesata din getMoney pentru a calcula comisionul la extragerea unei anumite sume din contul curent .
* SavingAccount are , la fel ca si clasa de mai sus , 4 metode asemanatoare ei : getDepAmount este metoda care returneaza cantitatea numerara aflata in cont in momentul accesarii interogarii soldului de catre client ; putMoney este asemanatoare metodei descrise mai sus , doar ca aceasta nu are o limita anume , aceasta metoda avand astfel doar un parametru , care se adauga la suma totala aflata in cont la momentul respectiv . Si aceasta metoda are ceva special , si anume , faptul ca persoana care detine contul va primi o suma mica in plus la fiecare introducere de bani in cont , o anumita dobanda ; getMoney este folosita , la fel ca si in metoda de mai sus , pentru a extrage bani , diferenta fiind faptul ca nu se mai calculeaza vreun comision , ci se pot extrage bani fara a fi modificata suma de bani de alte calcule .
* Clasa Person are 3 metode : getName prin care se intoarce numele persoanei , getCNP prin care se returneaza codul numeric personal si getAdress prin care se returneaza adresa .
* Clasa Account este asemanatoare clasei Person ca si numar de metode si rol al lor , astfel ca aceasta are tot 3 metode : getPIN prin care se intoarce pin-ul contului , getClient prin care se returneaza clientul cu toate caracteristicile lui . Acest client este de fapt un obiect de tip Person , astfel prin acesta se face legatura intre clasa Person si Account , legatura necesara , deoarece fiecare cont este detinut de catre o persoana si numai una ; metoda getType prin intermediul careia se returneaza tipul contului : SavingAccount sau SpendingAccount .
* Clasa Bank are 19 metode , dintre care una , isWellFormed() este folosita pentru assertiuni , adica in aceasta se verifica daca numarul elementelor din HashTable este la fel cu numarul elementelor din ArrayList-ul de conturi , utilizat pentru o accesare mai eficienta si rapida a datelor . Metoda createSavingAccount are un parametru de tip SavingAccount si este folosita pentru a crea un nou cont care este introdus atat in HashTable cat si in ArrayList-ul de conturi . Metoda createSpendingAccount este asemanatoare celei de mai sus , doar ca in aceasta metoda se creaza un cont de tip Spending Account . Si acesta va fi adaugat atat in HashTable cat si in ArrayList-ul de conturi . Metoda deactivateAccount primeste un String , care este de fapt pin-ul contului , si introduce acest pin intr-o lista de conturi dezactivate , astfel clientul nu isi va putea accesa contul pana nu va scoate acest pin din lista , adica sa isi reactiveze contul prin metoda urmatoare , activateAccount , care primeste tot un String ca si parametru , acesta fiind pin-ul ce trebuie scos din lista de de conturi dezactivate . Astfel , clientul va putea reintra in contul dezactivat si , daca vreo suma i-a ramas in acel cont , acesta va putea sa si-o refolosesca , adica banca nu ii fura banii din cont . Metoda searchByPIN este folosita pentru a cauta anumite conturi in functie de pin-ul lor , care este , de fapt , codul lor unic de indentificare si cheia pentru HashTable . notDeactivate este o metoda care returneaza true sau false , depinzand de faptul daca un cont a fost dezactivat sau nu . In aceasta metoda se cauta de fapt in ArrayList-ul de conturi dezactivate si se returneaza false daca pin-ul cautat nu s-a gasit printre acele elemente . searchTypeAcc returneaza tipul unui cont : Saving Account sau Spending Account . existPIN cauta pinul dat ca si parametru in lista de conturi si transmite true daca s-a gasit sau false in caz contrar . setActiveAccount este o metoda prin intermediul careia se schimba contul activ la un alt cont . Este folosita atunci cand se face transmiterea de bani in plus din Spending Account . Prin intermediul introducerii pin-ului pentru Spending Account se poate face schimbarea la acel cont , schimband doar variabila de activeAccount cu acel pin . Dupa ce se face depunerea de surplus in contul de Saving Account , se reseteaza contul la starea de dinainte , adica se reintroduce in activeAccount pinul precedent salvat in metoda de setActiveAccount . Acest lucrur se face prin intermediul metodei resetActiveAcc . showBalance este o metoda care face legatura cu contul de Saving Account pentru a putea transmite datelel despre suma de bani din acel cont . showBalanceATM este asemanatoare metodei anterioare , doar ca aceasta face legatura cu Spending Account si transmite suma de bani aflata in acel cont . extractMoney face legatura cu cu contul de Saving Account pentru a putea extrage bani din acel cont . . extractMoneyATM este asemanatoare metodei anterioare , doar ca aceasta face legatura cu Spending Account si extrage banii din acel cont . depositMoney primeste ca parametru o variabila de tip float si depune in contul de Saving Account o anumita suma transmisa prin acea variabila . depositMoneyATM este asemanatoare metodei anterioare , doar ca aceasta face legatura cu Spending Account , transmitand tot o anumita suma care va fi adaugata la suma deja existenta in acel cont .

Diagrama de clase :

♣ Modelează vocabularul sistemului ce trebuie dezvoltat ;

♣ Surprinde conexiunile semantice sau interacţiunile care se stabilesc între elementele componente ;

♣ Folosită pentru a modela structura unui program ;

Poate conţine :

♣ Clase/Interfeţe ;

♣ Obiecte ;

♣ Legaturi :

- Asociere

- Agregare

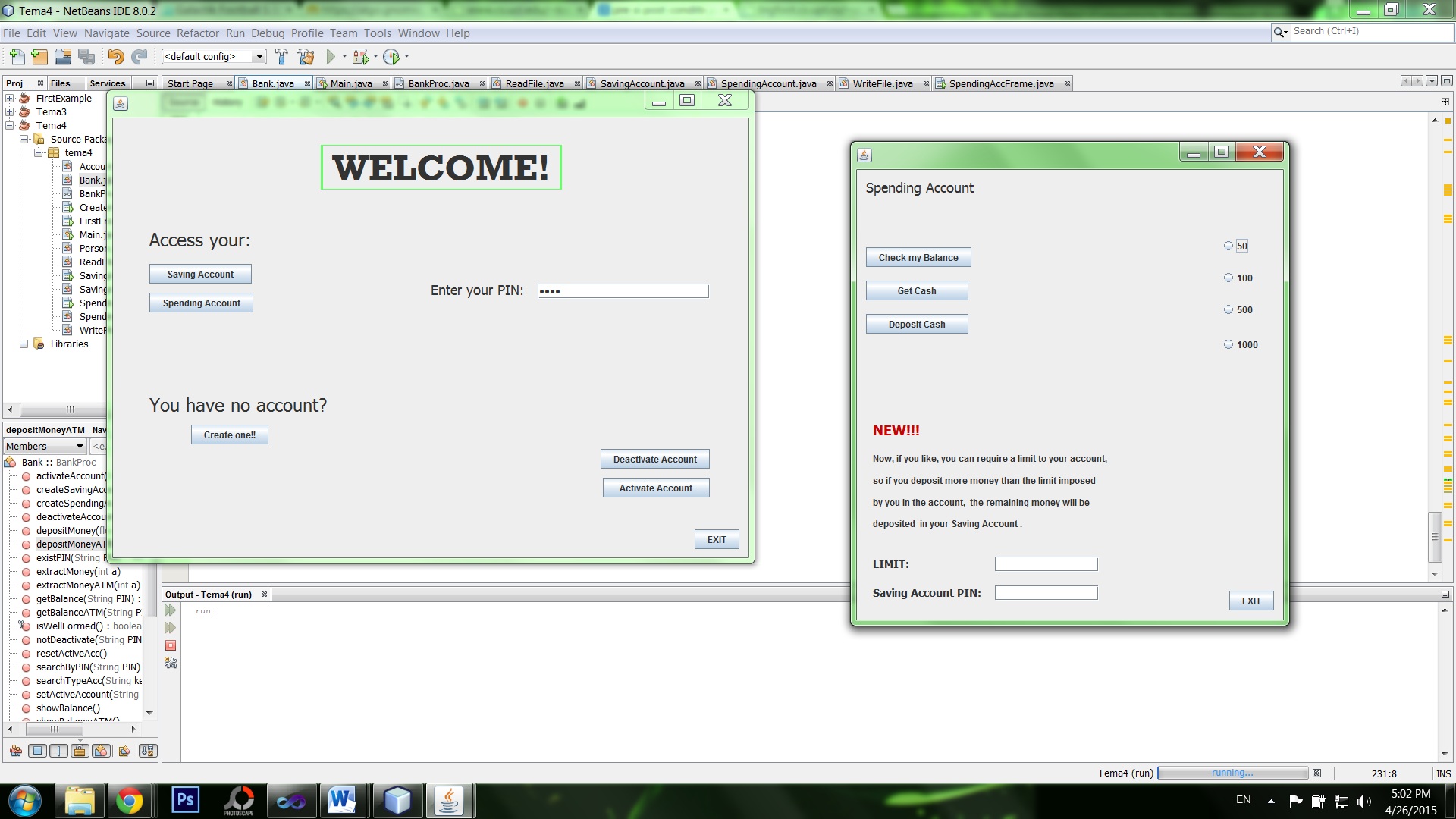
- Generalizare

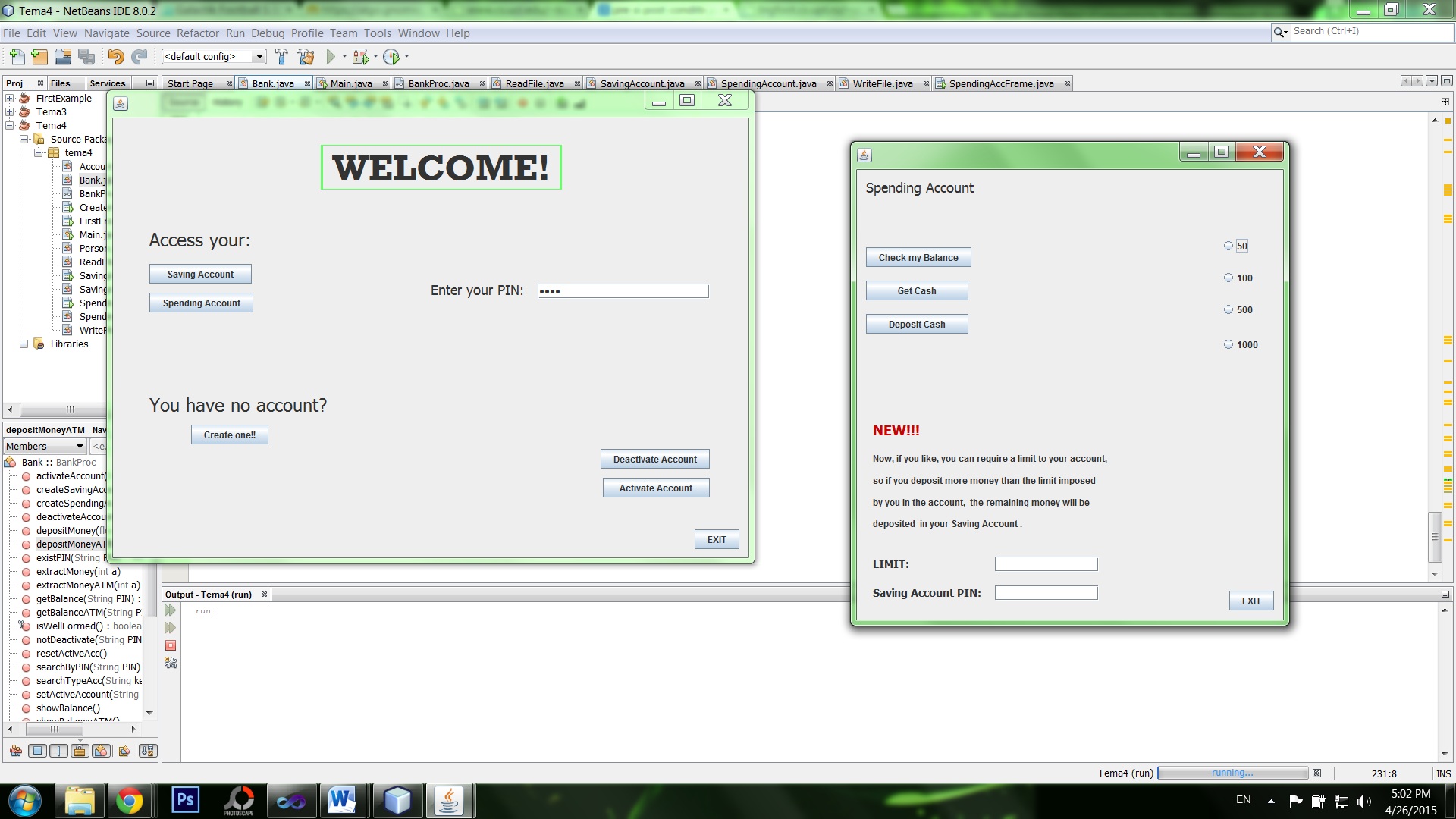
- Dependenţă

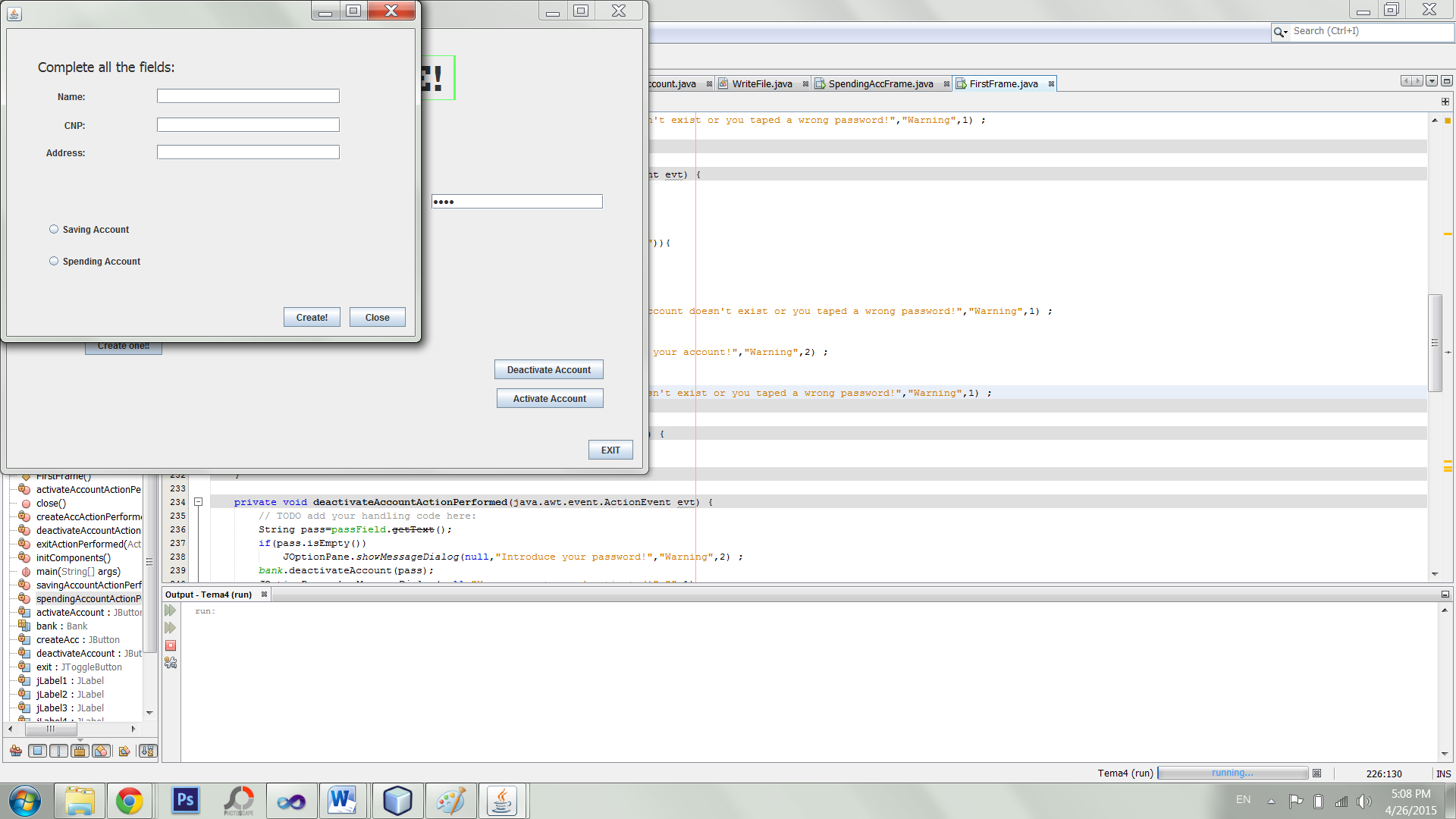
- Realizare

De aceea , am ales pentru proiect aceasta reprezentare a claselor , cu ajutorul diagramei de clasa UML .

* **3. Interfata Grafica**



****



Interfata grafica a fost realizata cu ajutorul programului NetBeans care dispunde de facilitatea de a realiza totul cu drag and drop si cateva click-uri . In realizarea ei am utilizat Table-uri pentru a le popula cu tabele din Baza de Date sau cu anumite date extrase de acolo cu ajutorul unor conditii ; am utilizat butoanele JButton de la Swing , JTextField si ComboBox-uri . Pentru intreaga fereastra se foloseste JFrame , iar pentru anumite portiuni ale Frame-ului , pentru a le face oarecum independente , se utilizeaza JPanel-uri . Pentru a putea introduce text in ferestrele mele am folosit JLabel . Pentru a putea beneficia de toate aceste obiecte , am importat javax.swing.\* .

JFrame este o versiune extinsă a clasei Frame care adaugă suport pentru un comportament de desenare special. Adiţional, JFrame permite componentelor Swing MenuBars să fie ataşate nu numai în partea de sus a ferestrei dar oriunde în fereastră. Toate obiectele asociate unui JFrame sunt manipulate de o instanţă a clasei JRootPane, care este singura componentă-fiu a unei instanţe JFrame. JRootPane este un container simplu pentru alte câteva componente.

Echivalentul Swing al clasei AWT Panel este JPanel. JPanel suportă toate tipurile de “layout manager” din AWT, plus cele noi din Swing. Clasa TestPanel este derivată din JFrame; creează o instanţă JPanel căreia i se aplică “GridLayout manager”. Butoanele sunt adăugate instanţei JPanel, ci nu ferestrei principale. O instanţă JPanel este implicit “double buffered”, ceea ce reduce efectul de “flicker” în Page | 5 timpul operaţiilor de redesenare a ecranului, pentru programele de animaţie. Dacă utilizăm “double-buffering” pentru o componentă, toţi fii acesteia vor utiliza de asemenea “doublebuffering” (chiar daca nu este activat). JRootPane este componenta din vârful ierarhiei oricărei ferestre Swing, deci activând “double-buffering” pentru JRootPane, toate subcomponentele sale vor fi desenate utilizându-se tehnica “double-buffering”.

Etichetele (JLabel) sunt string-uri de text care se utilizează pentru a identifica alte componente. Pot avea propriul tip de font, propriile culori “foreground” (culoarea textului) şi “background” (culoarea de fundal), şi pot fi poziţionate oriunde în containerul căruia îi aparţin.

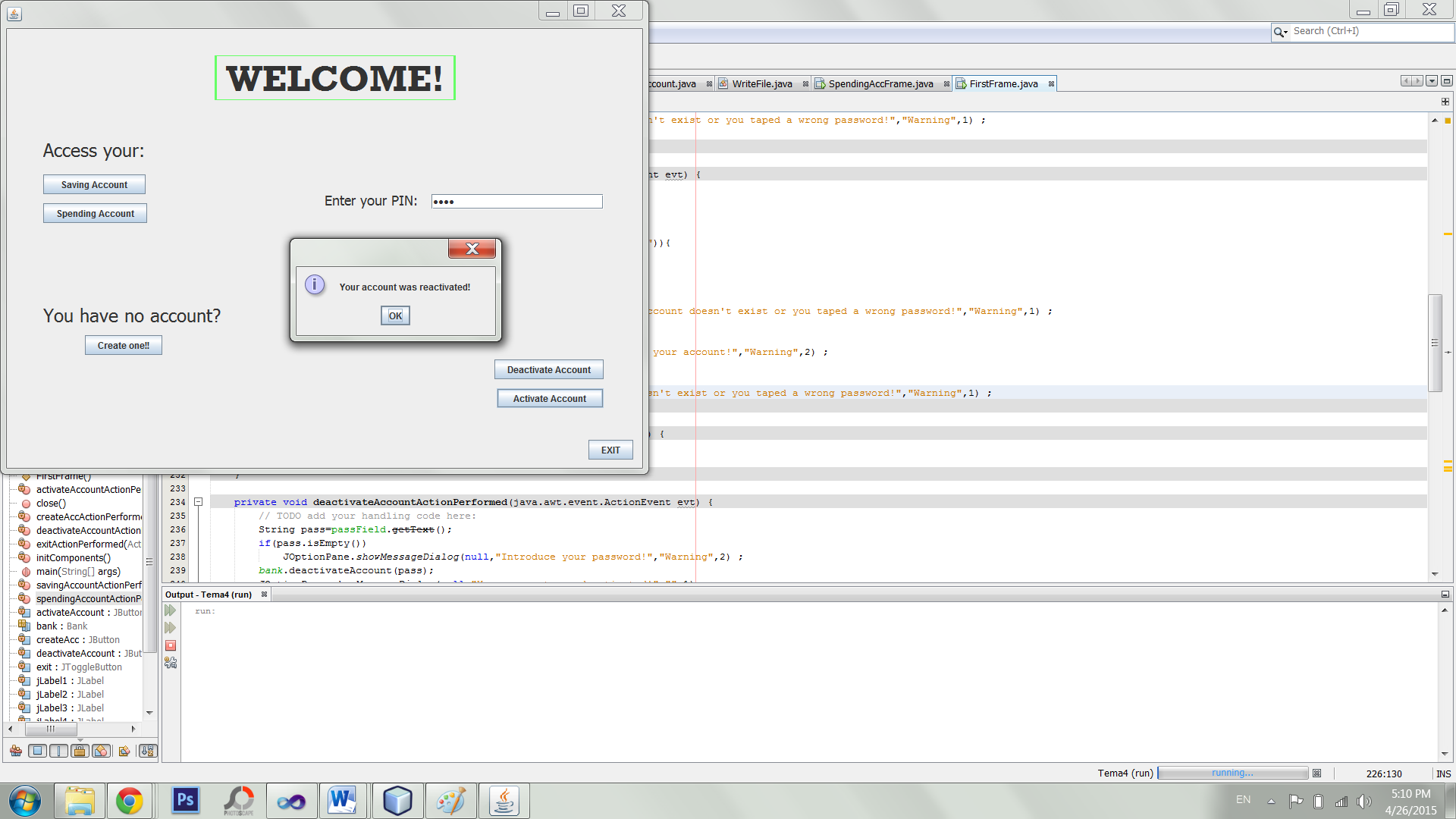
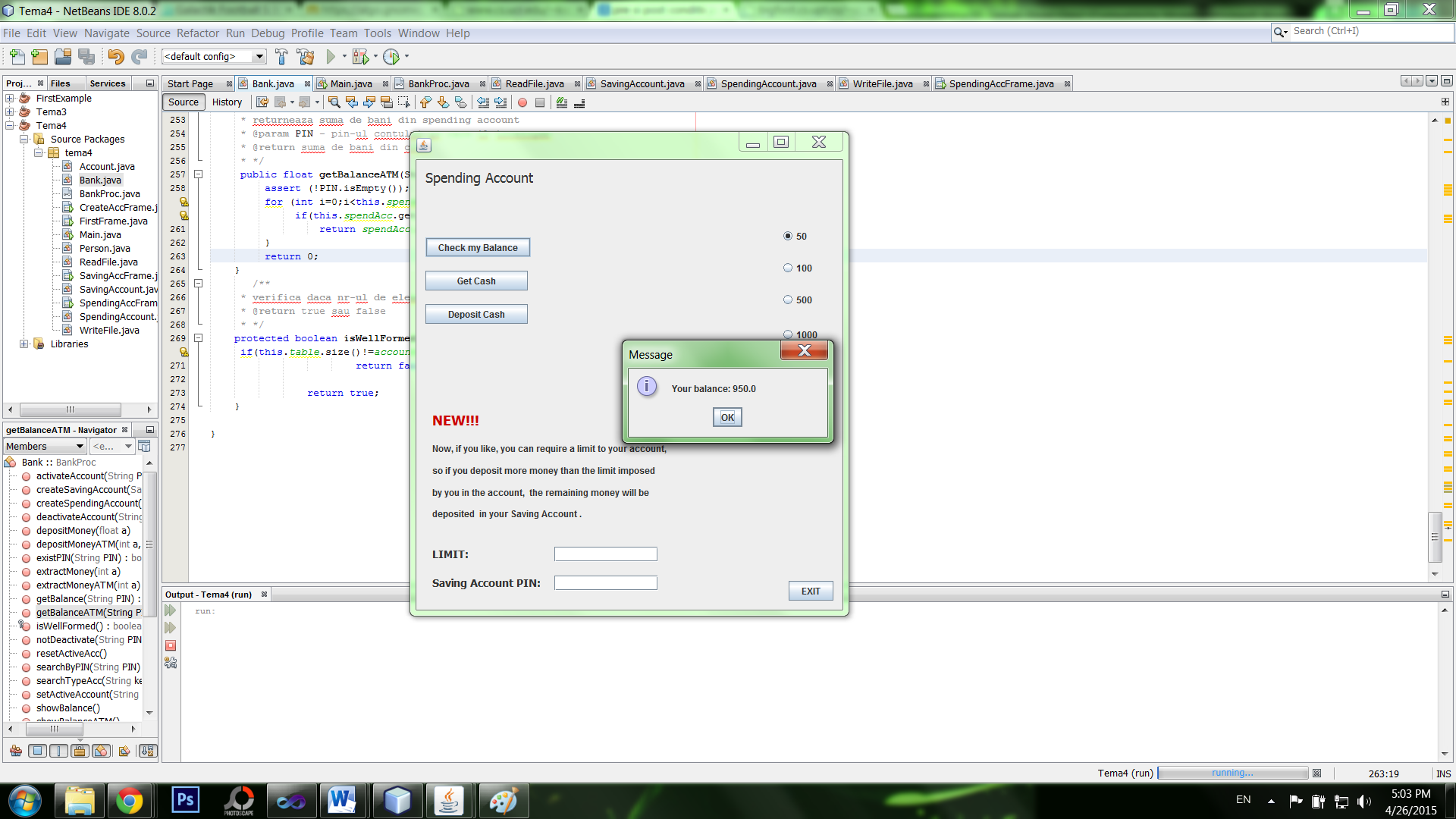
JButton sunt butoane a caror importanta este faptul ca au proprietatea de ActionListener, adica se poate creea un cod in spatele accesarii lor .

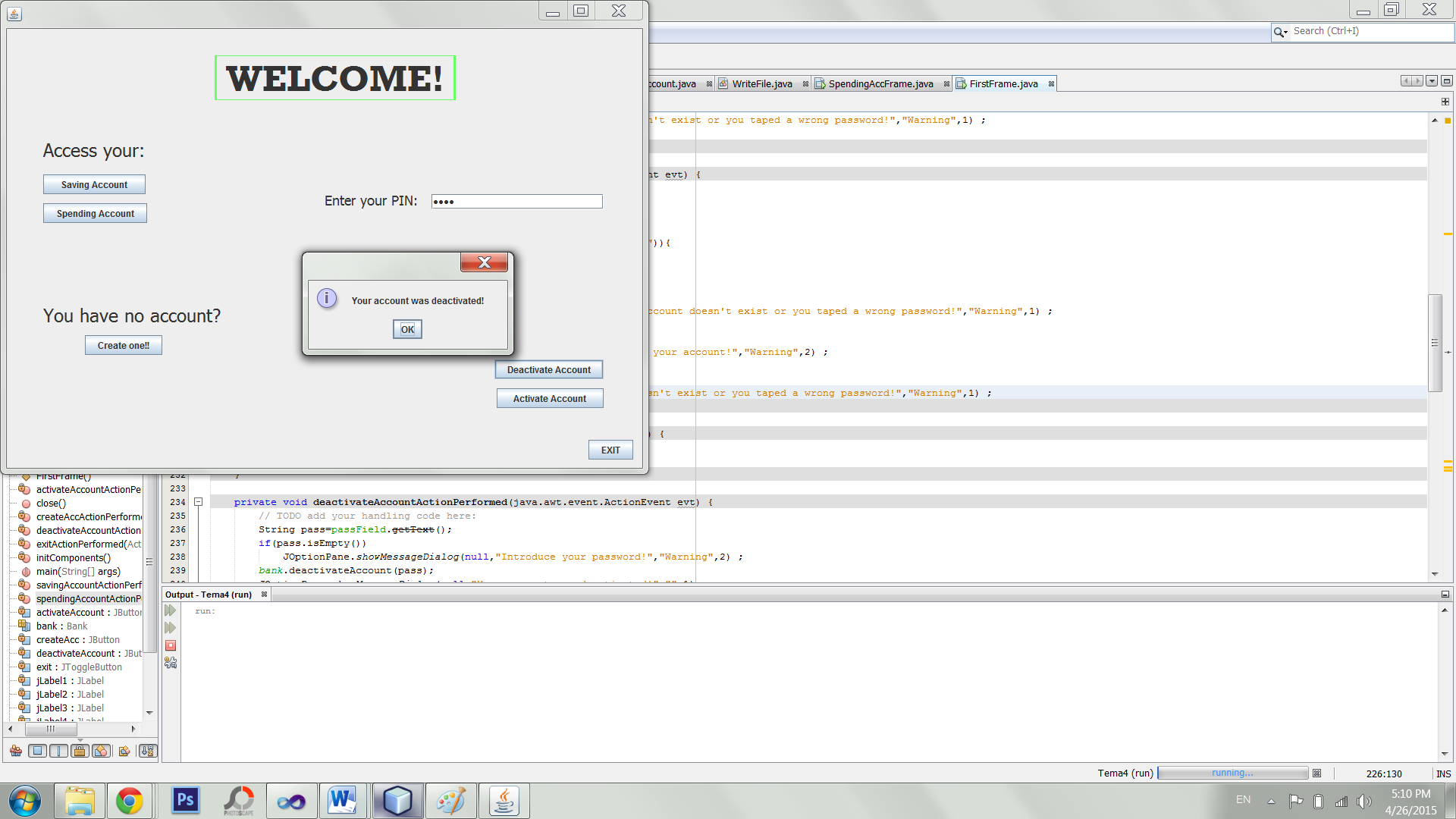
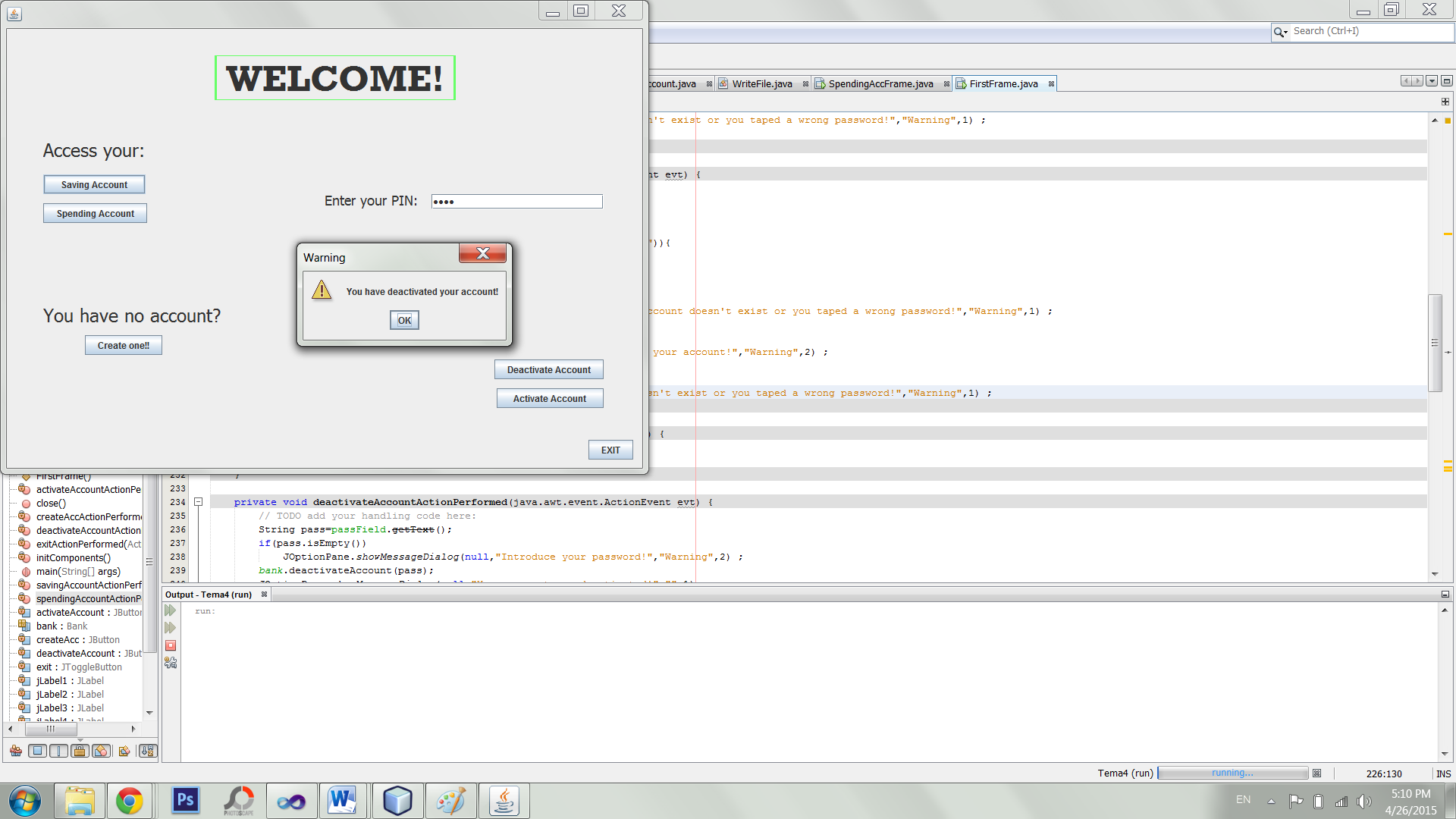
Clasa JComboBox permite utilizatorului să selecteze o opţiune dintr-o listă de opţiuni. Pentru a afla obiectul selectat se poate folosi metoda getSelectItem(), care returnează un Object sau getSelectedIndex(), care returneaza un ȋntreg. Pentru a seta o opţiune se foloseşte metoda getSelectedIndex(n), unde n este un ȋntreg care dă poziţia item-ului care dorim să-l selectăm.

Clasele JTextField şi JTextArea sunt componente care conţin text ce poate fi editat de utilizator. Un JTextField are o singură linie text, pe când JTextArea poate să aibă mai multe linii text. Se pot seta ambele să fie read-only, atunci utilizatorul poate doar să citească textul, nu să-l şi editeze. Ambele clase sunt subclase ale clasei abstracte JTextComponent.

**4. Implementare si testare**

Testarea a fost realizata cu ajutorul Debuger-ului si a testarii simple , prin care am introdus date si am observat ce exceptii imi sunt aruncate , incercand apoi sa le repar . Astfel , dupa testare , fiecare actiune gresita a utilizatorului in interactiunea cu interfata este penalizata doar cu un mesaj de avertizare , fara ca aplicatia sa dea crash .



  
****

**5.Concluzii. Ce s-a invatat. Dezvoltarii ulterioare.**

Implemenatea unui program care sa se ocupe de procesarea comenzilor nu este dificila din punct de vedere algoritmic .

In concluzie pot spune ca mi-a facut placere sa lucrez la aceasta tema avand o aplicabilitate practica importanta in zilele noastre .

Din perspectiva a ceea ce am invatat doresc sa mentionez ca mi se pare extrem de utila folosirea tabelelor de dispersie , iar prin aceasta tema am vazut cum se implementeaza si cum se poate folosi o astefel de tabela in Java . Pe langa acestea cred ca mi-am imbunatati considerabil abilitatile de a creea interfete grafice cu care se poate interactiona cat mai usor si care sa fie cat mai placute din punct de vedere estetic .

Aplicatia ar putea de asemenea sa foloseasca baze de date pentru a gestiona clientii si conturile

Dezvoltarea ulterioara a aplicatie poate cuprinde :

- sa fie dezvoltate conturi si pt alte monede $ , euro , etc .

- sa fie implementat un sistem de dobanzi si sa se vada cum dupa un anumit timp creste acea dobanda , astfel , dobanda sa se modifice in functie si de timp si de suma depusa la un anumit moment in cont ;

- sa se retina un istoric pentru cate extrageri sau facut sau cate depuneri

- sa se retina data pentru fiecare actiune si sa se ofere si o chitanta in format pdf de exemplu .

**6.Bibliografie**

* <http://download.oracle.com/javase/tutorial/> - tutorial java de la Sun
* „Java de la 0 la Expert” – Stefan Tanasa , Stefan Andrei , Cristian Olaru ; Editura Polirom
* Tutoriale YOUTUBE !