TEHNICI DE PROGRAMARE

TEMA 3

**Procesare comenzilor**

Gozar-Manu Ariana

Grupa 30225

An II - CTI (romana)

CERINTA:

Realizati o aplicatie software de tip OrderManagement care sa proceseze comenzile primite de la clienti. Aplicatia va trebui sa contina (minim) urmatoarele clase : Order , OPDept (Order Processing Departmet) , Customer , Product and Warehouse . Clasele OPDept and Warehouse folosesc BinarySearchTree (arbore binar de cautare) pentru a retine comenzile. Utilizarea arborelui de cautare binar poate fi inlocuita cu utilizarea unei Baze de Date din care sa se extraga datele necesare si sa se actualizeze , de asemenea .

Pe langa implementarea propriu-zisa a proiectului trebuie analizat domeniul aplicatiei pentru a determina structura si comportamentul claselor in vederea realizarii unei diagrame UML existinsa : 2 diagrame de clasa , una inainte si una dupa implementarea proiectului , o diagrama USE-case , 2-3 diagrame de secventa si o diagrama de activitate . Se cere utilizarea javadoc pentru documentarea claselor si testarea acestora cu JUnit sau TestNG .

Se vor intalni in implementare si anumite raportari : under-stock , over-stock , total, filtre, etc.

OBIECTIVUL TEMEI:

Ne propunem sa implementam un sistem software bazat pe ideea unui magazin in care se pot adauga produse de catre un administrator si se pot achizitiona de catre clienti. In spatele magazinului exista si un depozit , de unde i se vor transmite produsele clientului daca acestea nu exista in magazin , dar se afla in depozit . Acest lucru se va face daca , bineinteles , cumparatul va accepta sa ia produsul din depozit .

ANALIZA PROBLEMEI , MODELARE , SCENARII SI CAZURI DE UTILIZARE:

Ideea de baza a acestei aplicatii este intalnita in tot ceea ce inseamna comert in ziua de azi . Orice magazin este bazat pe aceasta idee . Aplicatia are o utilitate practica foarte ridicata luand in calcul ca cu trecerea timpului tot mai multi vor tine informatiile in format digital , chiar si la in minimagazine , deoarece se faciliteaza utilizarea dator , mai ales intr-o cantitate ridicata .

Aplicatia trebuie sa simuleze atat ceea ce se intampla din punctul de vedere al utilizatorului - client , cat si din punctul de vedere al administratorului . Lucrul cel mai important pe care aplicatia il utilizeaza si care trebuie luat in considerare la realizarea acestei aplicatii este introducere si scoaterea datelor din Baza de Date , functie de anumite conditii . Cel mai important aspect de care sunt strans legate aceste conditii este utilizarea cheii primare la conceperea Bazei de Date , care ne ofera posibilitatea de a accesa fiecare tupla in mod unic . Introducerea in depozit si magazin a produselor priveste doar administratorul care le ofera spre vanzare , interfata pentru acesta fiind prima componenta a interfetei grafice realizate pentru aplicatie .A doua componenta este partea de interactiune a clientului cu magazinul . Acesta poate atat sa cumpere , cat si sa-si vada cumparaturile realizate , sa manipuleze unele date aflate pe profilul lui si sa vada anumite actualizari facute de administrator care il privesc in mod direct . Toate aceste activitati se intampla in timp real .

Modelarea problemei de rezolvat implica stabilirea volumului datelor de intrare si anume:

- produsele dunt descrise prin id , nume , pret , stoc . Acestea pot fi adaugate in magazin si depozit , iar pentru cele existente se pot face schimbari , de stoc , de exemplu .

- un client isi poate adauga intr-o comanda orice produs din magazin si depozit . Acesta acceseaza depozitul doar daca nu gaseste un anumit produs in magazin , dar il gaseste in depozit si astfel este intrebat daca doreste achizitionarea lui de acolo , acesta putand raspunde prin DA sau NU . El poate efectua mai multe comenzi , totul rezumandu-se la existenta a ceea ce cauta el in magazin sau depozit .

Diagrama Use – case

Utilizator

Administrator

Descrierea diagramei :

Actori : Administratorul si Clientul

Sistem: Depozit si magazin – propunere spre vanzare si achizitionare

Descriere scenariu:

* Administratorul introduce produsele in depozit sau magazin. Acesta poate sa modifice stocul doar pentru unele produse , nu sa le mai adauge , duplicandu-le .
* Administratorul poate cauta dupa stoc produse , pentru a il actualiza , lucru foarte util cand stocul pentru anumite produse este 0 .
* Administratorul poate sa vada utilizatorii inscrisi pentru magazinul sau si sa stearga dintre ei , insa nu se poate sterge pe el .
* Administratorul actualizeaza starea comenzilor functie de cum sunt acestea : in procesare, trimise sau fara vreo stare anume .
* Un client poate sa-si achizitioneze produse in limita stocului disponibil .
* Daca doreste o cantitate prea mare , atunci va fi anuntat ca nu exista cantitate dorita de acesta .
* Atunci cand nu este multumit de o comanda , clientul o poate anula doar daca aceasta nu a fost trimisa .

PROIECTARE:

UML anterior :

|  |
| --- |
| **Order** |
|  |

|  |
| --- |
| **OPDept** |
|  |

|  |
| --- |
| Warehouse |
|  |

|  |
| --- |
| **Customer** |
|  |

UML dupa proiectare:

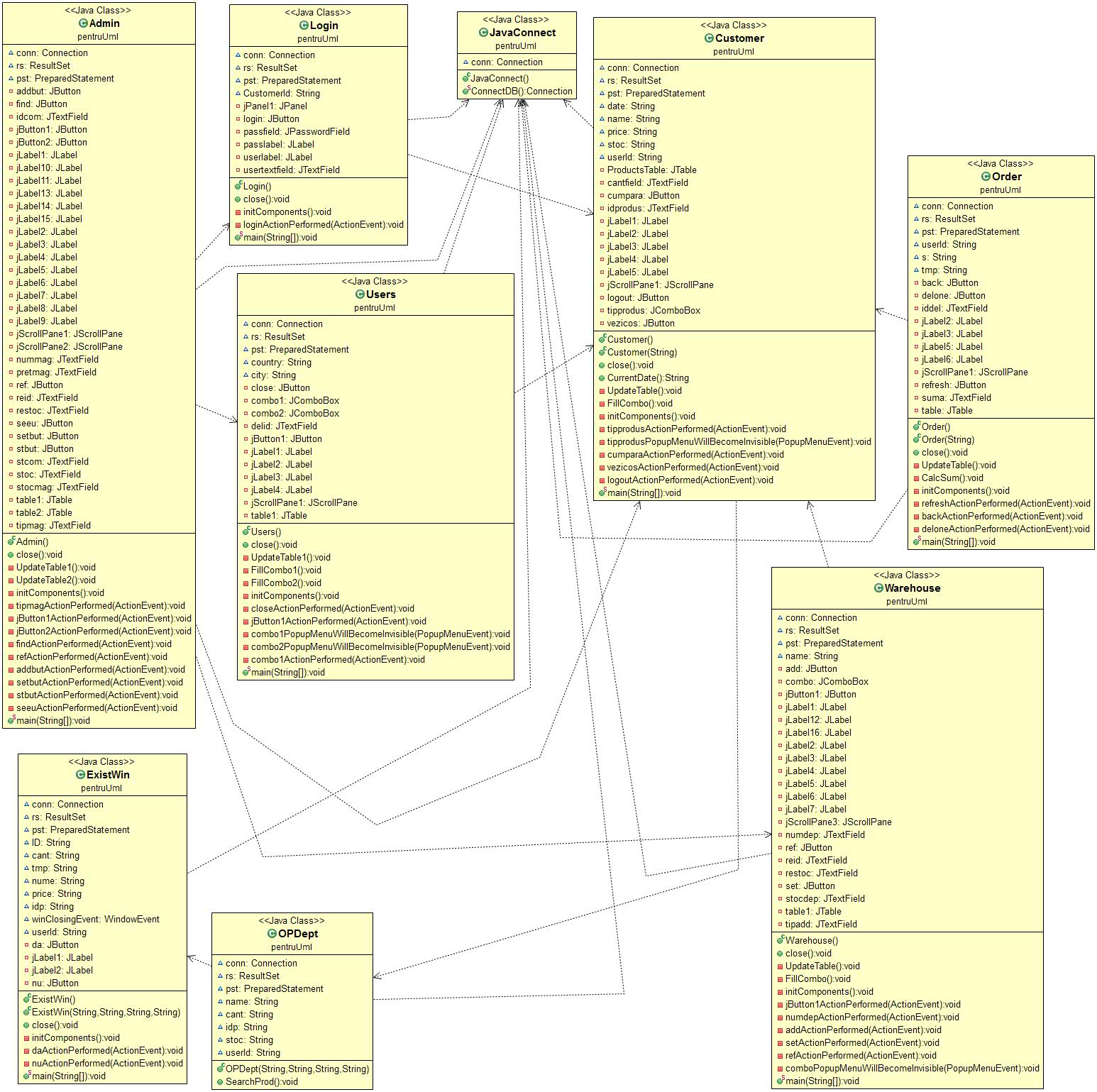


Diagrama UML s-a complicat foarte mult deoarece am intrpdus un administrator care are grija de introducerea produselor in depozit si in magazine si de actualizarea anumitor lucruri . De asemenea , atunci cand un produs este gasit in depozit , deoarece nu exista in magazin , clientului ii apare un noua frame in care este intrebat daca doreste sau nu achizitionarea produslui , astfel ia nastere o noua clasa ExistWin . Clasa Users este folosita ca pentru a realiza partea de interfata in care clientul poate sa cumpere un produs , pe cand clasa Customer va fi folosita pentru a realiza partea de interfata in care clientul isi vede comenzile si le poate schimba in anumite cazuri . Pentru o realizare cat mai sugestiva a acestui magazin , am realizat si o casuta de logare pentru fiecare utilizator in parte . Clasa JavaConnect este necesara pentru a face conexiunea cu Baza de Date .

Diagrame de secventa:

cumparareProdus(ID)

cumparareProdus(ID)

DB - abstractizare

Customer

X

Customer

DB - abstractizare

Login

searchFor(user,password)

returnMessage(“User si password incorecti!”)

searchFor(user,password)

returnMessage(“User si password corecti!”)

setVisible(Customer)

returnMessage(“Produs inexistent!”)

populeazaTabel(resultSetQuery())

returnQueryResult()

sendConfirmMessage()

updateDB()

sendMessage(“DA”)

sendMessage(“DA”)

showMessageToChoose()

sendMessage(“Exista”)

SearchInDepozitProdus(ID)

ShowMessage(“Nu exista!”)

SearchInMagazinProdus(ID)

DB - abstractizare

ExistWin

OPDept

Customer

setVisible(ExistWin)

sendConfirmMessage()

NU

DA

Diagrama de activitate:

DA

Clase si metode utilizate:

Clasa Admin este realizata pentru a oferi Administratorului o interfata de interactiune cu aplicatia , astfel aceasta apare doar atunci cand in aplicatie utilizatorul se logheaza cu Admin ca si username si admin ca si parola . Metodele aflate in aceasta clasa se refera la partea de magazin si cea de comenzi , existand buton de legatura cu clasa Users si Warehouse , de unde administratorul poate accesa datele referitoare la depozit si clientii magazinului pe care il administreaza si le poate manipula. Exista doua metode , UpdateTable1 si UpdateTable2 , prin care se populeaza cele doua tabele aflate pe interfata grafica cu date aduse din Baza de Date referitoare la totate comenzile facute de clientii magazinului si toate produsele din acesta . Exista doua metode prin care contin codul pentru delogare si reimprospatare a paginii , foarte utila atunci cand se produc schimbari printre produse si comenzi . De asemenea exista o metoda findActionPerformed in care se cauta in Baza de Date toate produsele cu stocul egal cu cel introdus de utilizator si se afiseaza acele produse in tabela ce tine de magazin . In cele doua metode setbutActionPerformed si stbutActionPerformed se seteaza anumite caracteristici , si anume , stocul unor produse si statusul comenzilor create de clienti . In metoda addbutActionPerformed se creaza legatura cu Baza de Date pentru a insera produse in magazin . Pentru ca administratorul sa vada si datele referitoare la depozit si utilizatori , exista doua metoda prin care se fac vizibile frame-urile pentru manipularea datelor .

Clasa ExistWin are doi constructori deoarece aceasta poate fi accesata si fara parametrii, dar si cu ajutorul a patru stringuri ce semnifica date despre produsul cautat in depozit . Exista si o metoda destul de comuna tuturor claselor , si anume metoda close() , cu ajutorul careia se inchide frame-ul nou deschis la apasarea unui buton ales , pentru aceasta Clasa fiind atat butonul DA cat si butonul NU . La apasarea butonului DA se acceseaza metoda daActionPerformed si se fac schimbarile necesare atat in Baza de Date cat si in interfata destinata comenzilor clientului pentru ca se inregistreaza o cumparare a unui produs . Daca se apasa NU , fereastra se va inchide si niicio schimbare nu va fi efectuata .

Clasa OPDept este accesata pentru a cauta in depozit un anumit produs . Daca se gaseste, cu ajutorul metodei SearchProd() , se va face vizibila partea de interfata implementata in Clasa ExistWin si se vor produce schimbari functie de alegerea clientului . Constructorul acestei clase are patru stringuri care contin date referitoare la produsul cautat . Daca nu se va gasi un produs in depozit se va returna un mesaj sugestiv .

Clasa Login implementeaza partea de interfata care ne apare prima data cand rulam aplicatia . In acele campuri destinate introducerii numelui utilizator si a parolei , uitilizatorul trebuie sa scrie date valide pentru a puta accesa mai departe aplicatia deoarece se face legatura cu Baza de Date si se cauta mai intai daca exista numele de utilizator in tabela Customer , iar dupa aceea se cauta daca la numele de utilizator introdus ii corespune parola introdusa . Daca nu se potriveste , atunci va fi afisat un mesaj de eroare si utilizatorul poate reincerca sa introduca alte date sau poate sa inchida aplicatia . Daca s-au introdus date valide , se va inchide aceasta fereastra si se va deschide fie fereastra destinata clientului , fie cea destinata administratorului , daca utilizatorul a folosit username-ul si parola pentru a se loga ca si administrator .

JavaConnect este o clasa fara prea mult cod , insa cu o foarte mare importanta , deoarece toate celelalte clase intra in legatura cu ea pentru a putea intra in legatura cu datele stocate in Baza de Date . Aceasta are o singura metoda , ConnectDB , in care se utilizeaza metodele consacrate in Java pentru conexiune. Programul utilizat pentru a creea Baza de Date este un Add-on de la Mozzila Firefox , numit sqlite . Acesta e usor de utilizat si ofera suportul necesar pentru a putea crea proiecte mai micute .

Clasa Customer implementeaza interfata grafica cu care se intalneste clientul dupa introducerea datelor de logare potrivite . Pe aceasta fereastra acesta va observa o tabela populata cu produsele magazinului . Acest lucru este realizat cu ajutorul metodei UpdateTable , utilizata in fiecare clasa care implementeaza o interfata ce contine un tabel . De asemenea , in interfata intalanim si un ComboBox , cu ajutorul caruia clientul poate filtra produsele , alegand tipul dorit pe care doreste sa-l vizualizeze . Pentru a popula acel ComboBox se utilizeaza metoda FillCombo . Atunci cand se alege un tip din fereastra care se va extinde atunci cand accesam ComboBox-ul se acceseaza metoda tipprodusActionperformed in care se executa o interogare referitoare la aducerea datelor cu tipul ales de client din Baza de Date . Tot in aceasta metoda se repopuleaza tabelul petru a arata acum produsele de un anumit tip . La apasare butonului Cumpara , se acceseaza metoda cumparaActionperformed , in care se cauta produsul in magazin dupa identificatorul sau unic . Daca nu se gaseste , se apeleaza clasa OPDept care cauta produsul in depozit . Daca nu se gaseste , se trimite un mesaj corespunzator , altfel , clientului i se lasa posibilitatea de a alege daca doreste sau nu sa cumpere produsul aflat in depozit . Daca clientul doreste sa-si vada produsele achizitionate , el poate accesa butonul de Vezi Cosul , accesand astfel metoda vezicosActionPerformed in care doar se face vizibila fereastra ce contine datele despre comenzi . Pentru a iesi din magazin , se poate face logout si se va reveni la casuta de logare .

Clasa Order contine metode pentru a manipula comenzile pentru fiecare client in parte , adica depinde de ID-ul utilizatorului care s-a logat , de aceea unul din constructori are un parametru cu un string , prin care se trimite acel ID . Cu ajutorul interfetei implementate in aceasta clasa , clientul poate sa-si manipuleze comenzile , si anume , sa-si stearga unele dintre ele . Daca se va incerca stergerea comenzilor care au statusul Sent , se va observa ca acest lucru este imposibil . Se pot sterge doar acele comenzi care nu au un status sau au statusul Procesing.. . Dupa ce s-a sters o comanda , tabelul nu va fi actualizat automat , ci numai dupa apasarea butonului Refresh , prin care se va accesa metoda refreshActionPerformed in care se va apela metoda UpdateTable . La stergerea comenzii se acceseaza metoda deloneActionPerformed in care se executa o interogare prin care se sterge datele despre comanda transmisa prin identificatorul sau unic din Baza de Date . Pentru a opri aceasta fereastra in care clientul isi poate observa comenzile si manipula , se apasa butonul Back , apelandu-se din metoda specifica lui metoda close() de inchidere a ferestrelor , specifica tuturor claselor care implementeaza o interfata , inafara de Login . O alta metoda importanta din aceasta clasa este CalcSuma() in care se calculeaza totalul pe care clientul il are de platit pentru comanda lui prin inmultirea cantitatii cu pretul fiecarui produs , iar dupa aceea realizarea sumei intre ele .

In clasa Warehouse , se implementeaza interfata cu ajutorul careia , utilizatorul logat ca si administrator , poate accesa date din depozit si modifica . Si aici intalnim metoda UpdateTable prin care se populeaza un tabel cu toate produsele aflate in depozit si datele despre aceastea . Exista si un ComboBox prin care se poate alege ce tip de produs doreste administratorul sa vada . Metodele gasite in aceasta clasa se refera la manipularea produselor aflate in depozit . Se pot adauga produse , se poate seta stocul unui produs sau se pot afisa tprodusele clasificate in tipuri . Toate aceste tipuri de metode s-au mai intalnit si sunt descrise in celelalte clase modul in care se realizeaza . Dupa realizarea unei manipulari , datele noi aflate in depoit nu vor fi vizibile pana nu se apasa butonul de Refresh in care , ca si in celelalte clase , se apeleaza metoda UpdateTable() .

Ultima clasa a aplicatiei este Users in care se implementeaza o interfata pe care o poate vizualiza doar administratorul deoarece sunt afisati toti utilizatorii inregistrati in magazin si se pot sterge o parte din ei . Singurul utilizator care nu se poate sterge este administratorul . In aceasta clasa exista doua metode de umplere a ComboBox-urilor deoarece exista doua ComboBox-uri . Unul este destinat filtrarii clientilor in functie de tara din care provin , iar celalalt este destinat filtrarii clientilor in functie de oras . Stergerea se realizeaza cu ajutorul identificatorului unic aflat in tabela pentru fiecare client . Aceasta fereastra se poate inchide prin apasarea butonului Inchidere pentru a reveni in fereastra de administrator implementata in clasa Admin .

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.\* .

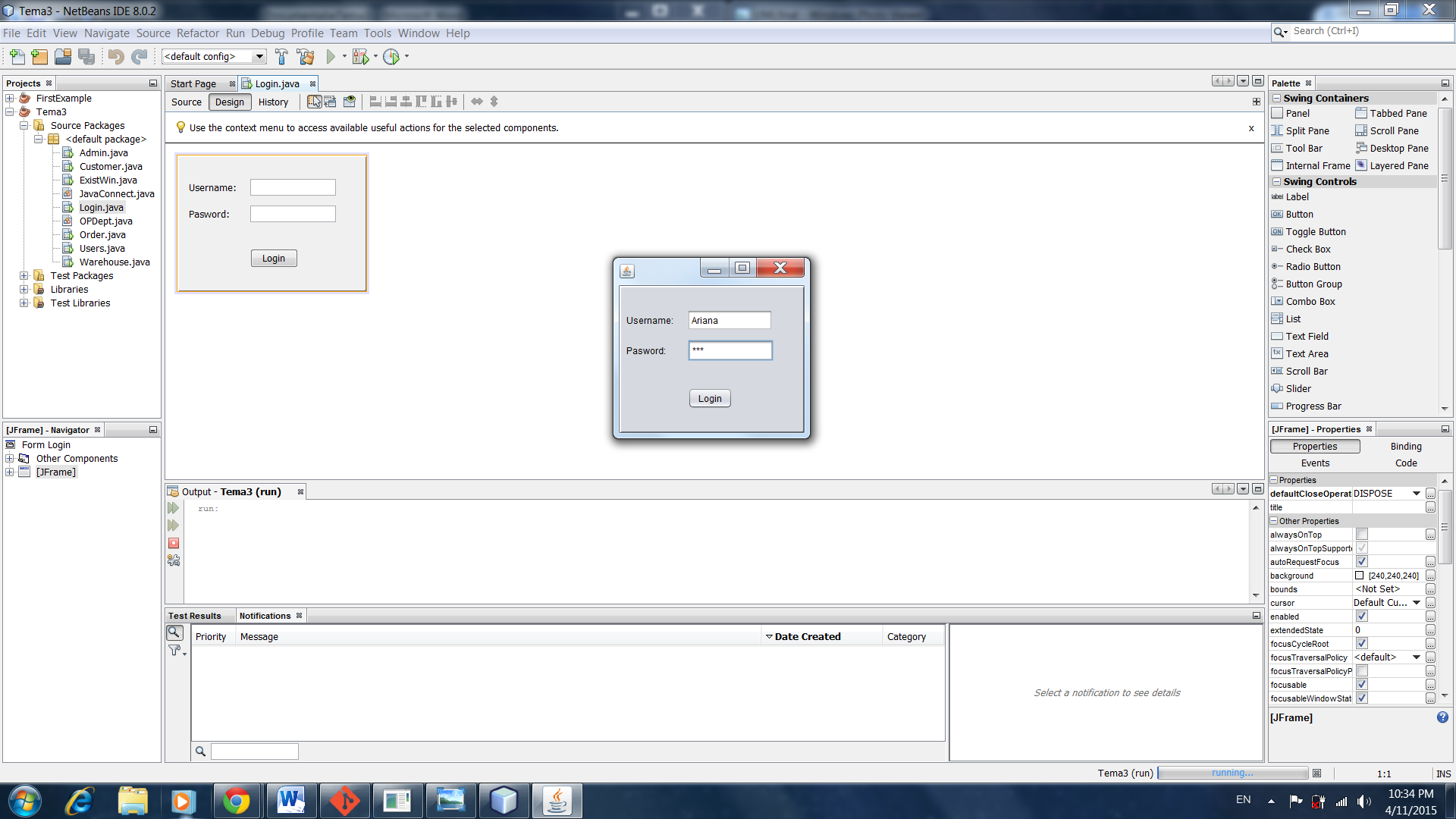
JFrame este o versiune extinsă a clasei Frame care adaugă suport pentru un comportament de desenare special. Adiţional, JFrame permite componentelor Swing MenuBars să fie ataşate nu numai în partea de sus a ferestrei dar oriunde în fereastră. Toate obiectele asociate unui JFrame sunt manipulate de o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

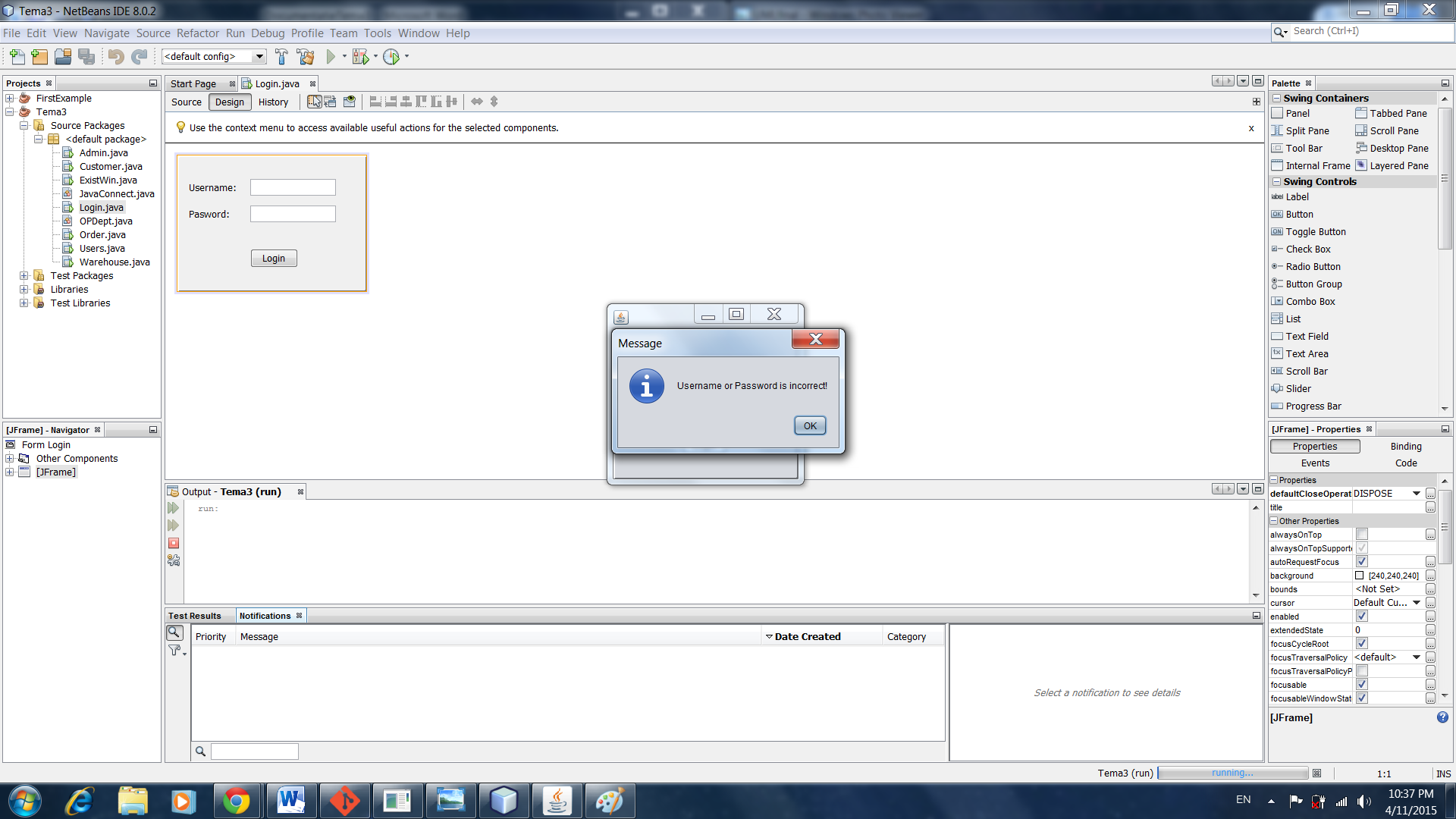
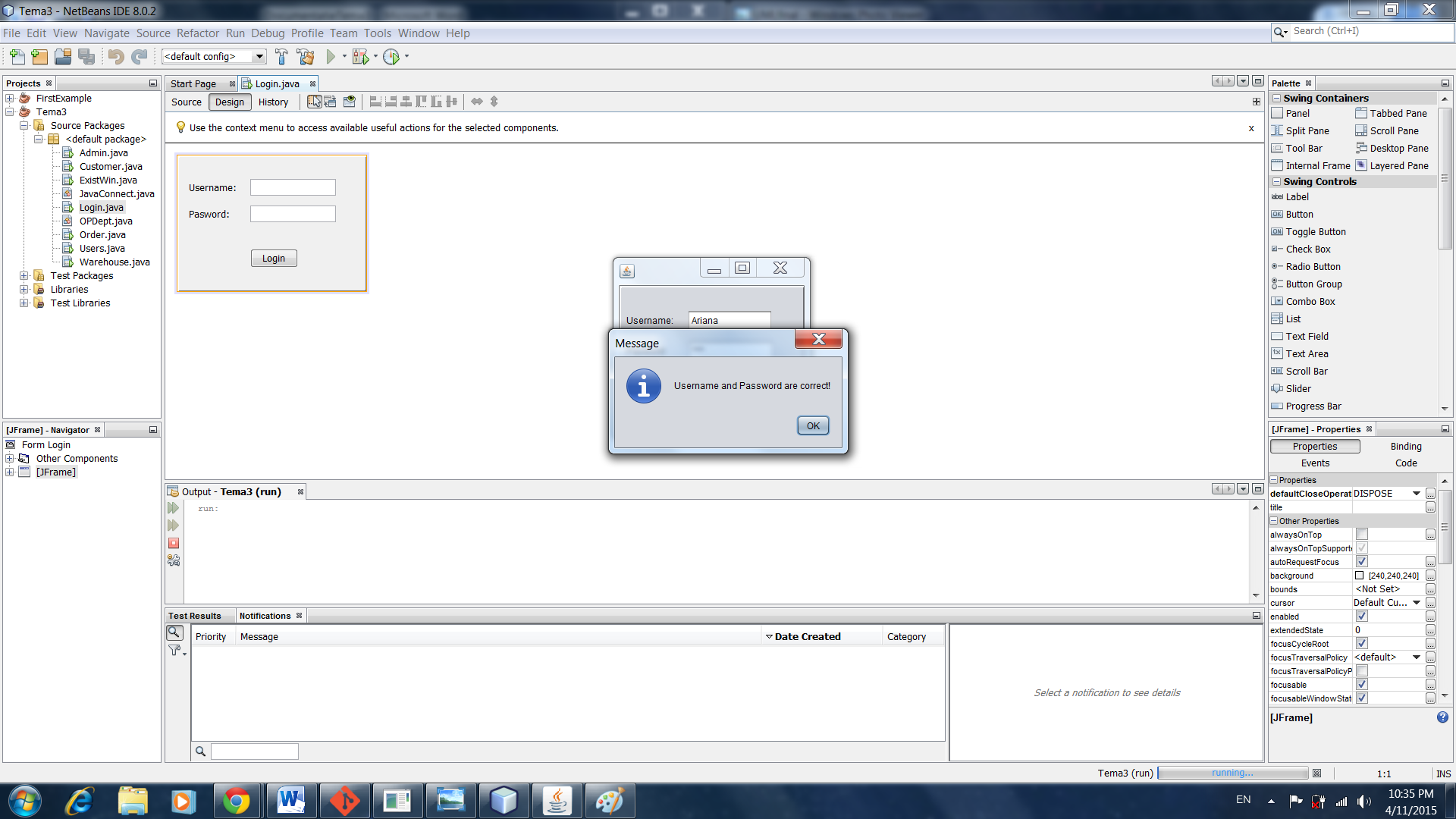
Echivalentul Swing al clasei AWT Panel este JPanel. JPanel suportă toate tipurile de “layout manager” din AWT, plus cele noi din Swing. Clasa TestPanel este derivată din JFrame; creează o instanţă JPanel căreia i se aplică “GridLayout manager”. Butoanele sunt adăugate instanţei JPanel, ci nu ferestrei 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”.

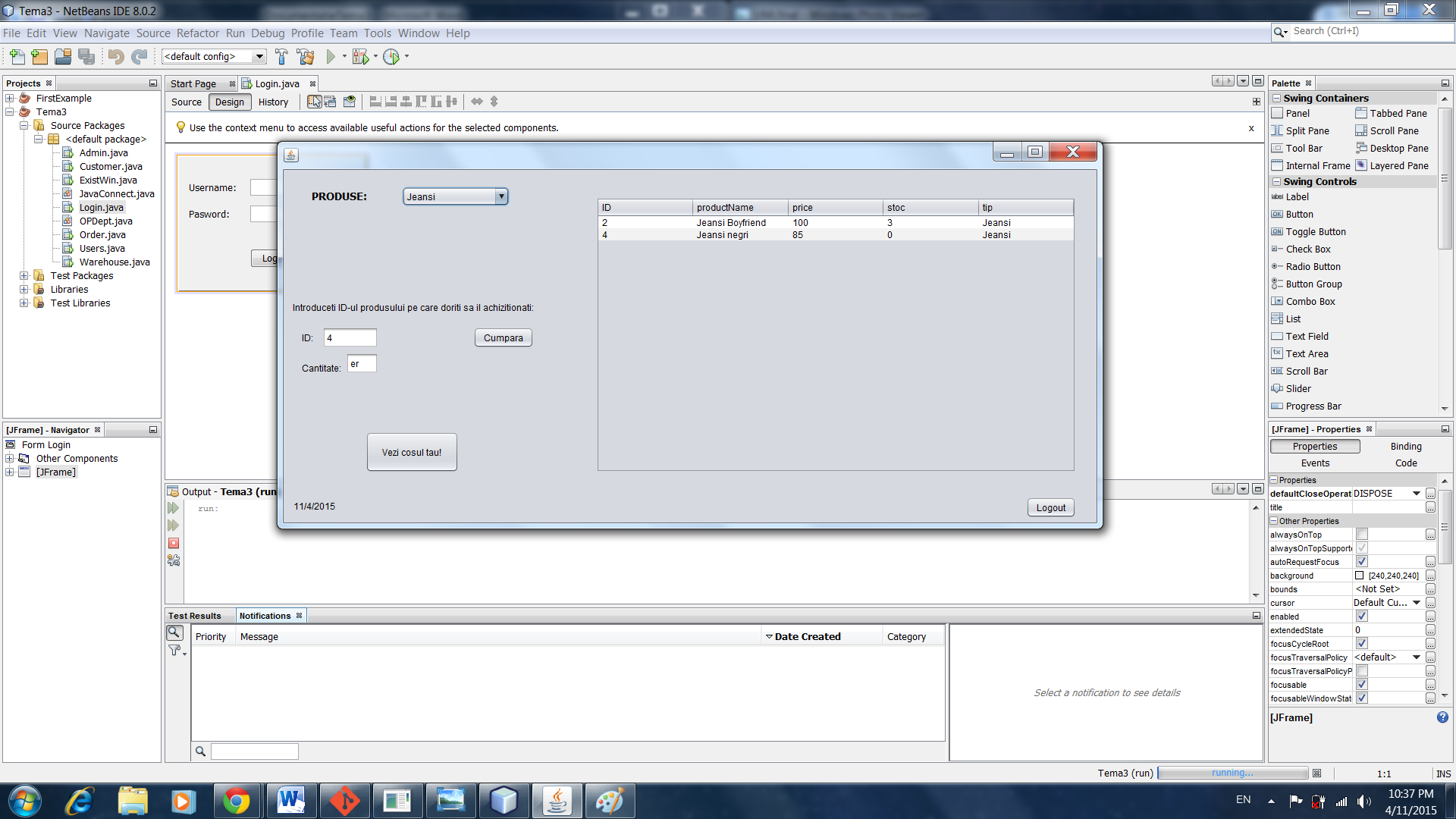
Etichetele (JLabel) sunt string-uri de text care se utilizează pentru a identifica alte componente. Pot avea propriul tip de font, propriile culori “foreground” (culoarea textului) şi “background” (culoarea de fundal), şi pot fi poziţionate oriunde în containerul căruia îi aparţin.

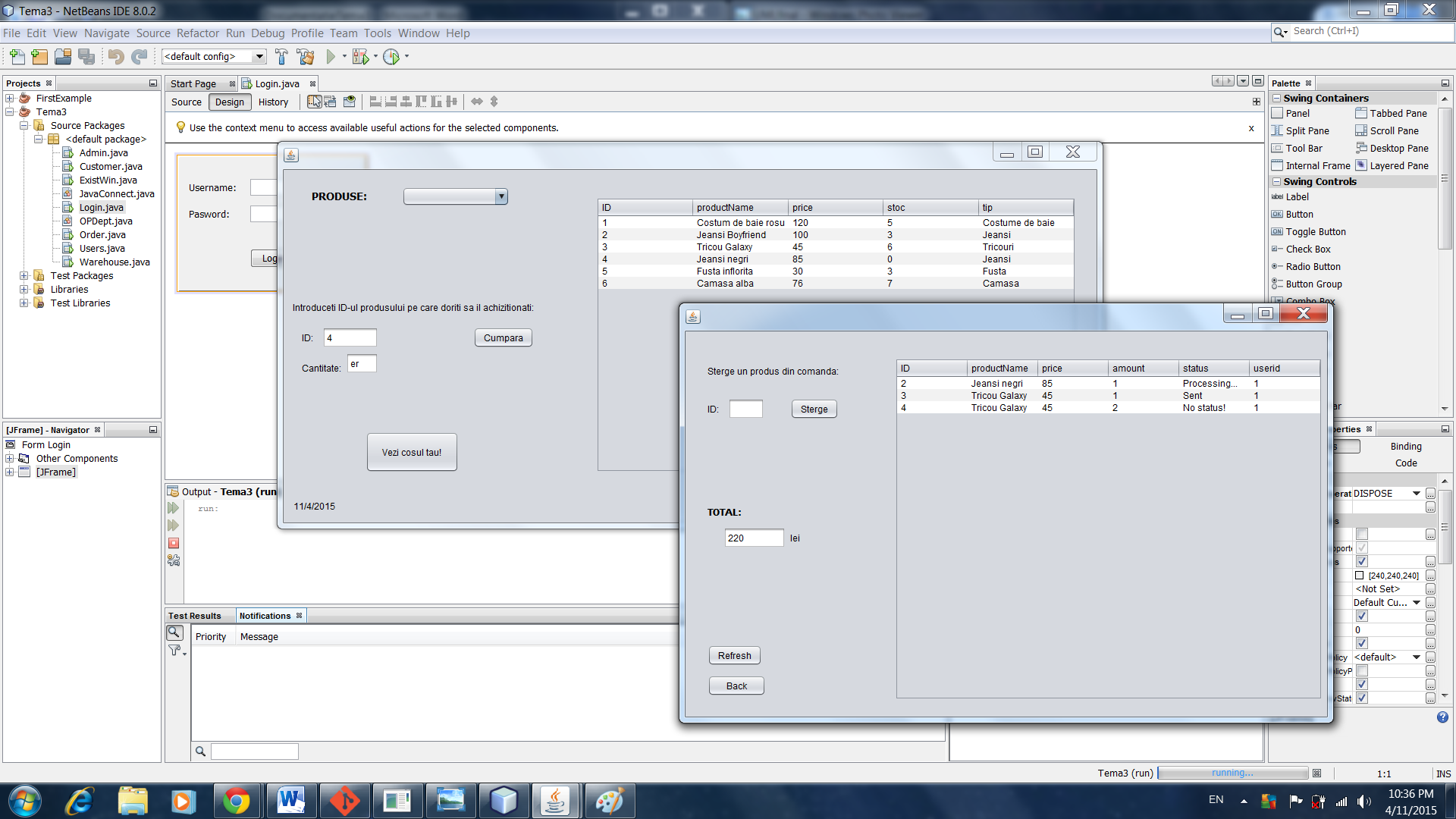
JButton sunt butoane a caror importanta este faptul ca au proprietatea de ActionListener, adica se poate creea un cod in spatele accesarii lor .

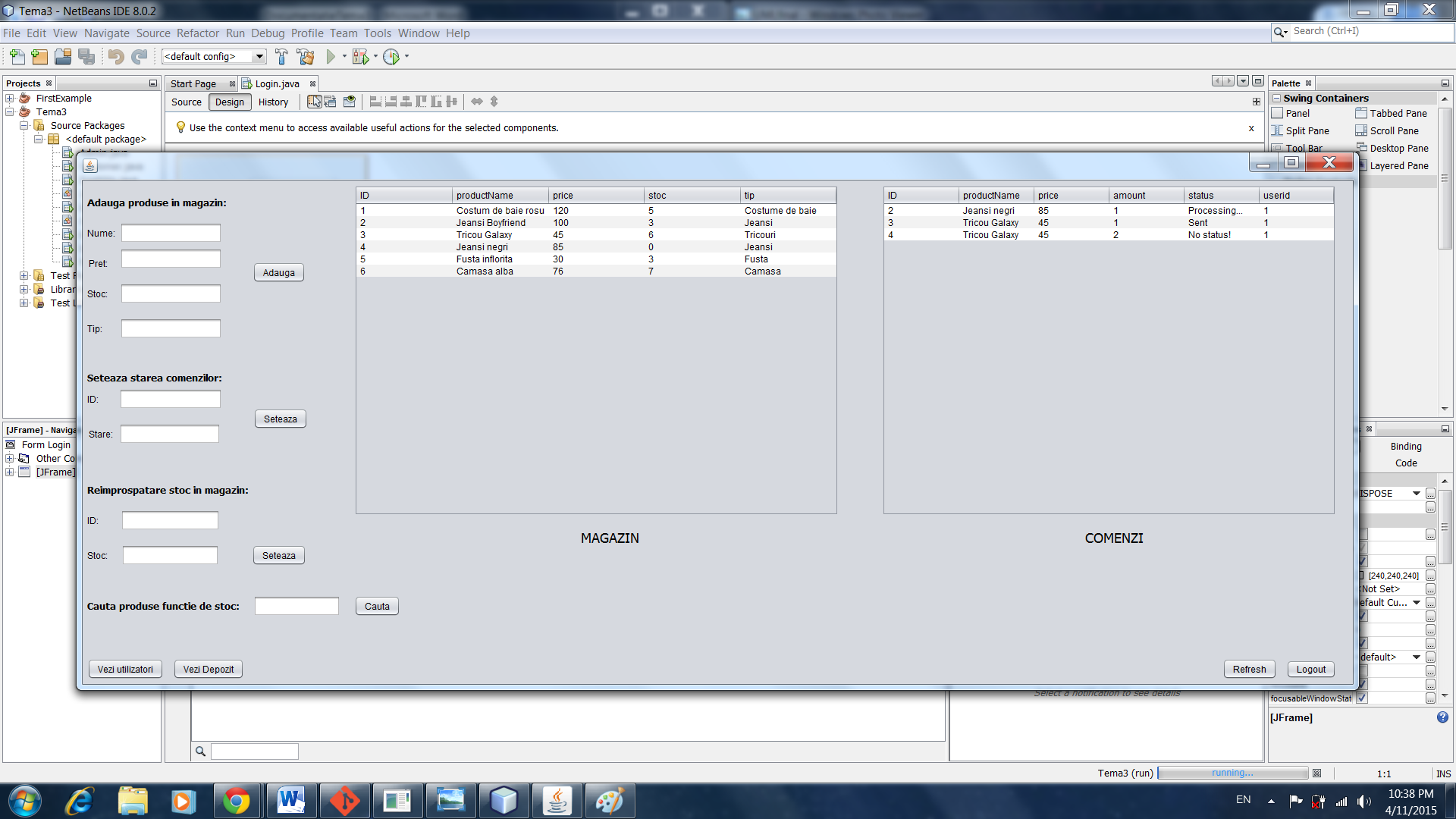
Clasa JComboBox permite utilizatorului să selecteze o opţiune dintr-o listă de opţiuni. Pentru a afla obiectul selectat se poate folosi metoda getSelectItem(), care returnează un Object sau getSelectedIndex(), care returneaza un ȋntreg. Pentru a seta o opţiune se foloseşte metoda getSelectedIndex(n),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

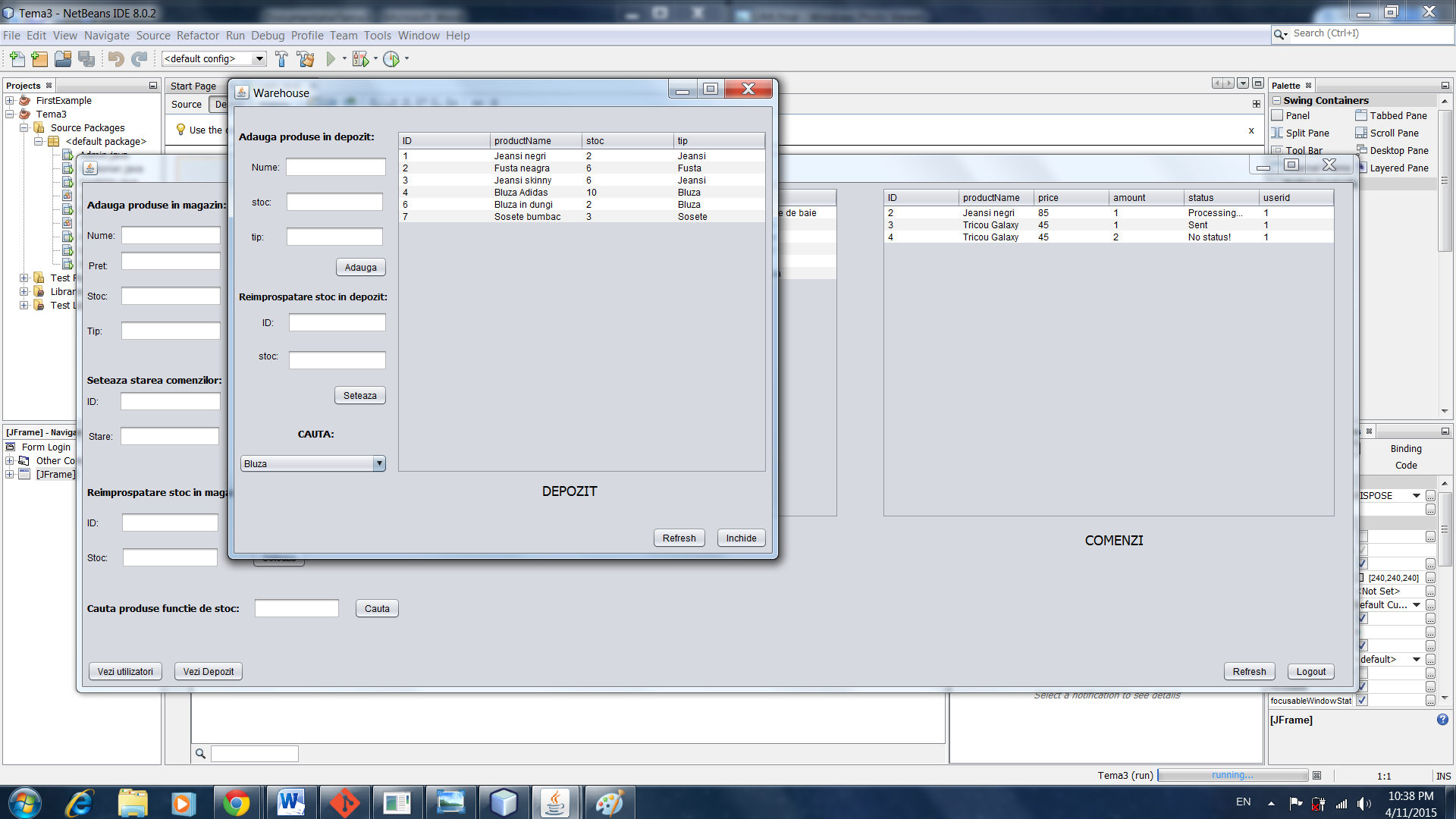
 Clasele JTextField şi JTextArea sunt componente care conţin text ce poate fi editat de utilizator. Un JTextField are o singură linie text, pe când JTextArea poate să aibă mai multe linii text. Se pot seta ambele să fie read-only, atunci utilizatorul poate doar să citească textul, nu să-l şi editeze.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

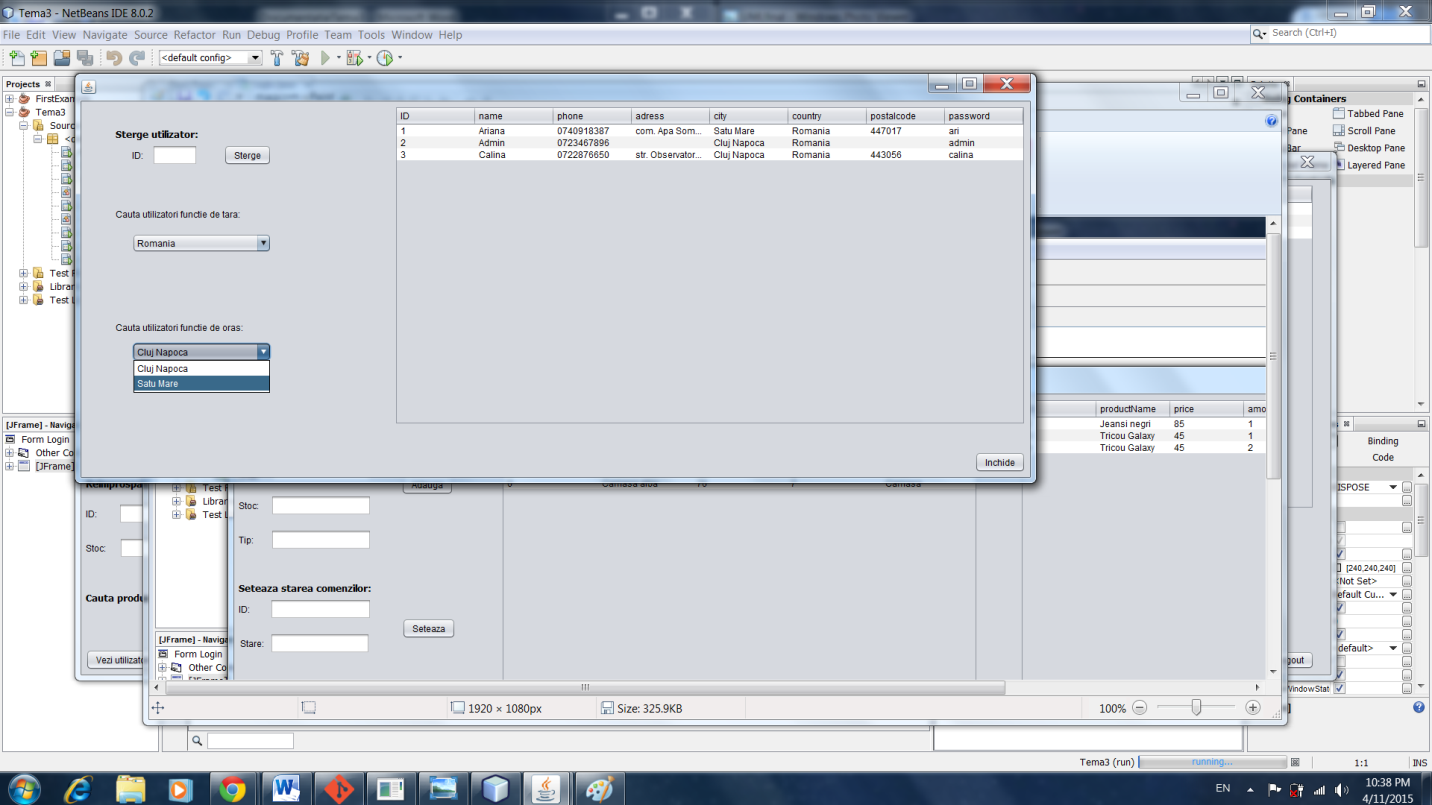


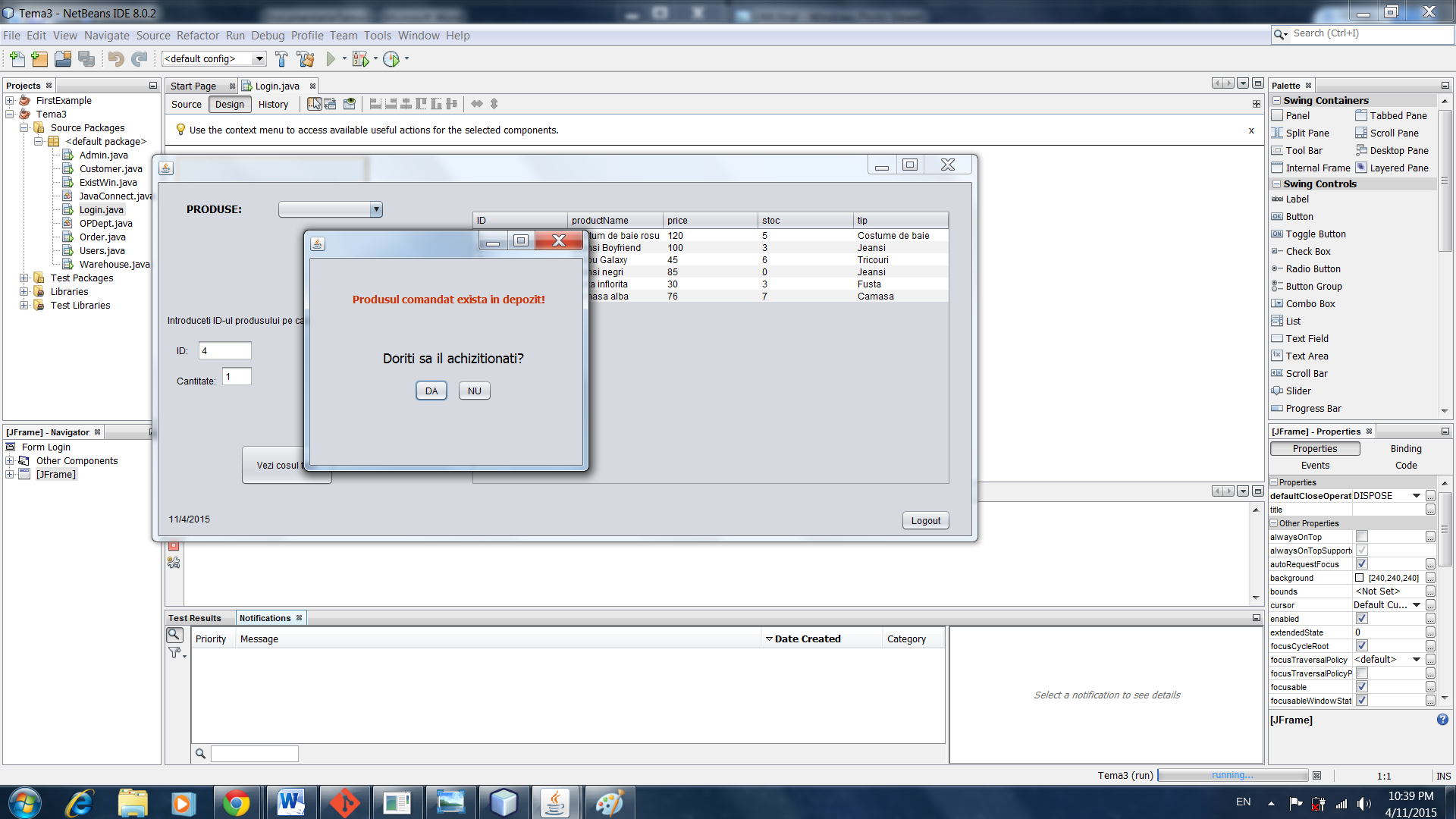












TESTARE:

Testarea s-a realizat cu ajutorul JUnit Test-ului pe care NetBeans il are deja inclus . Acesta realizeaza un sablon pentru fiecare clasa in parte si pentru fiecare metoda cu ceea ce s-ar putea testa si in ce mod .

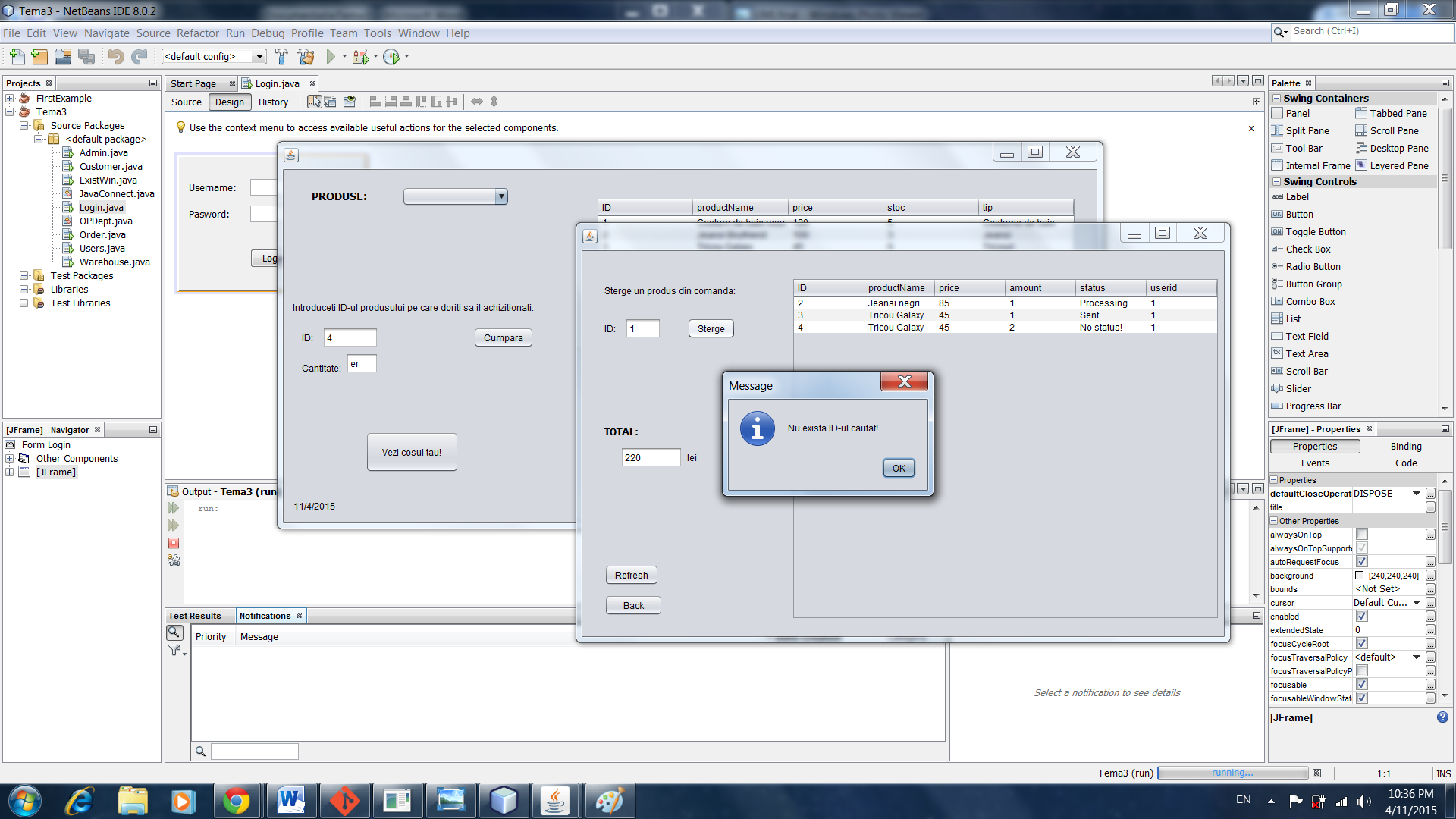
Testarea unitară s-a impus în ultima perioadă în dezvoltarea proiectelor scrise în limbajul Java şi numai, pe măsura apariţiei unor utilitare gratuite de testare a claselor, care au contribuit la creşterea vitezei de programare şi la micşorarea drastică a numărului de bug-uri .

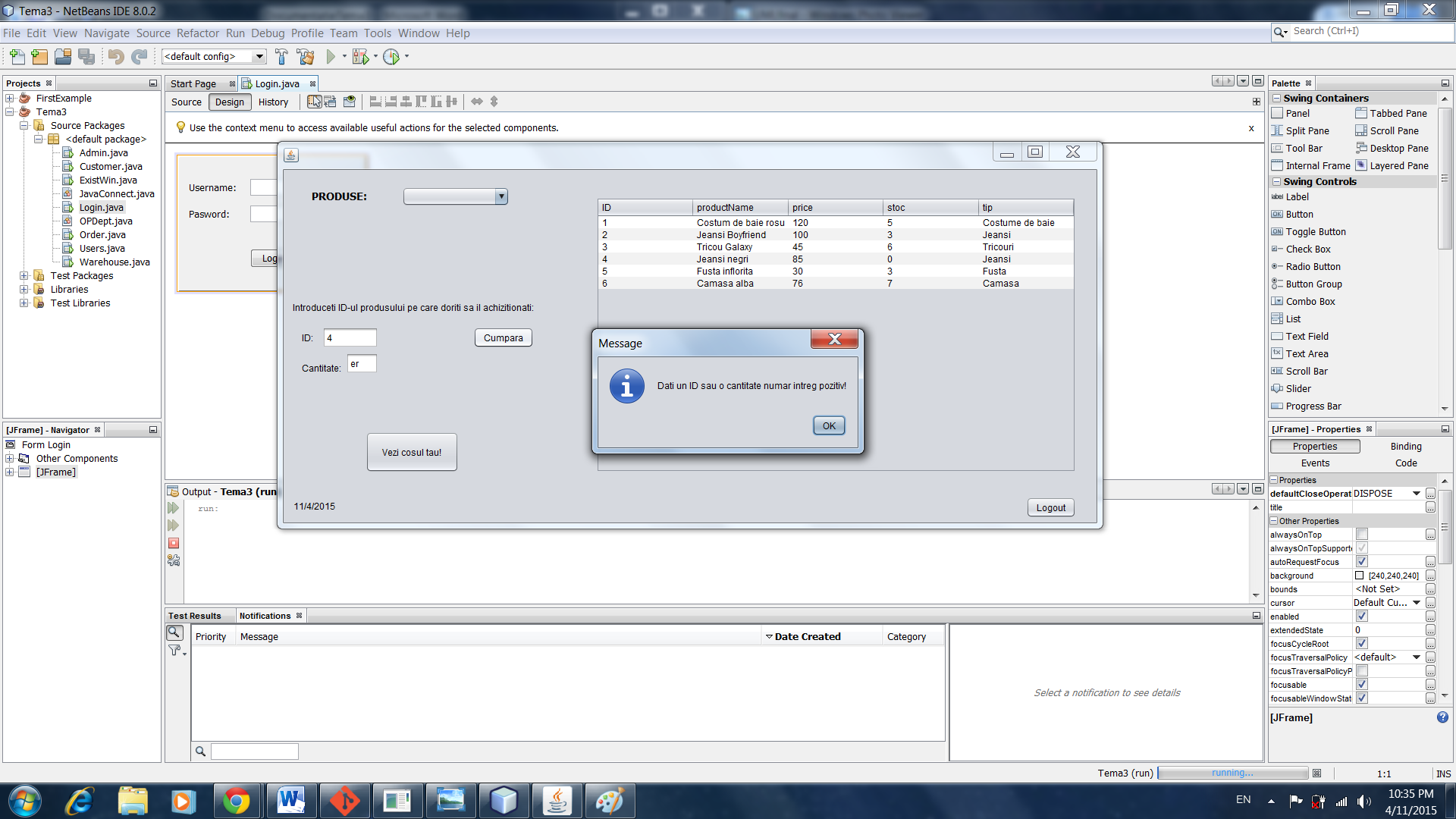
Printre avantajele folosirii utilitarului JUnit se numără:

* se îmbunătăţeşte viteza de scriere a codului, concomitent cu creşterea calităţii acestuia, deoarece prin scrierea testelor unitare se micşorează timpul de depanare, pemiţând refactorizarea mai uşoară, cu depistarea imediată a eventualelor erori inserate în codul modificat;
* clasele de test sunt uşor de scris şi modificat pe măsură ce codul sursă se măreşte, putând fi compilate împreună cu codul sursă al proiectului. Compilatorul testează sintaxa codului sursă, în timp ce clasele de test validează integritatea codului
* clasele de test JUnit pot fi rulate automat (în suită), rezultatele fiind vizibile imediat. Se pot crea ierarhii de suite de test, care pot fi testate împreună sau separat, în funcţie de cerinţete proiectului
* clasele de test măresc încrederea programatorului în codul sursă scris şi îi permit să urmărească mai uşor cerinţele de implementare ale proiectului, putând constitui şi o parte a documentaţiei finale transmise clientului
* JUnit este un utilitar gratuit, iar testele JUnit sunt scrise în Java şi beneficiază de portabilitatea acestuia

JUnit a fost proiectat pe baza a două modele (patterns): modelul Command si modelul Composite.

De asemenea, testarea a fost realizata si 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 .





DEZVOLTARI 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.

Aplicatia se poate mari , prin introducerea unor noi metode de manipulare deoarece , in opinia mea , in acest domeniu al realizarii aplicatiilor bazate de Baza de Date , limita de complexitate tinde la infinit . Scopul temei a fost sa ne familiarizam cu utilizarea Baxei de Date in realizarea de aplicatii , de aceea nu am incercat sa fac proiectul foarte complex ci doar sa implementez anumite lucruri de baza .

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

• YOUTUBE!