UNIVERSITATEA POLITEHNICA BUCURESTI

FACULTATEA DE ELECTRONICA, TELECOMUNICATII SI TEHNOLOGIA INFORMATIEI

Proiectarea interfetelor cu utilizatorul UI

Colesnicencu Claudiu
Popa Marian-Gabriel
Visan Valentin
Grupa 441 A

Bucuresti 2013

CUPRINS

Introducere in Java si GUI

Java este un limbaj de programare obiect orientat, cu scop general, concurent, bazat pe clase, proiectat sa aibe cat mai putine dependinte de implementare. Este creat cu scopul de a lasa dezvoltatorii de aplicatii sa “scrie o data, ruleze oricand” (“write once, run anywhere” – WORA), adica codul care ruleaza pe o platform nu trebuie recompilat pentru a rula sip e alta. Aplicatiile java sunt compilate in bytecode (fisiere .class) care pot rula pe orice masina virtuala Java (JVM) indiferent de arhitectura calculatorului. Java este unul dintre cele mai populare limbaje de programare folosite, in special pentru aplicatii web client-server.

Limbajul isi trage sintaxa din C si C++, dar are mai putine facilitati de nivel jos (“low-level”) decat oricare dintre cele doua.

Pentru a evidentia detasarea limbajului de programare Java fata de “parintele” acestuia, putem face cateva mici comparatii :

In timp ce C ofera programare procedural, functionala, obiect orientate si metaprogramare pe template-uri , Java incurajeaza o paradigma de programare obiect-orientata. C permite apeluri directe catre librariile native ale sistemului, in timp ce Java permite apeluri prin Java Native Interface si mai recent prin Java Native Access, librarii specializate care initial limiteaza accesul la librariile native ale sistemului pentru a proteja datele de programatorii mai putin experimentati.

Java GUI/Swing

Majoritatea aplicatiilor astazi folosesc o interfata grafica, ce are importantul rol de comunicare intre utilizator si computer. Diferenta intre lucrul cu o consola pentru folosirea unei aplicatii software si o interfata grafica este aceea ca interfata grafica interactioneaza in mod direct cu utilizatorul si creeaza astfel un mediu “user-friendly”. Orice aplicatie software ce are ca destinatie finala un utilizator oarecare (fara experienta in domeniu IT) trebuie sa foloseasca obligatoriu o interfata grafica (GUI – Graphic User Interface) deoarece aceasta este mult mai usor de invatat chiar si pentru cineva care nu a avut tangente cu programul respectiv.

De cele mai multe ori aplicatiile sunt perceput de utilizatorii finali doar prin intermediul interfetelor acestora si de aceea o interfata prost realizata poate compromite intreaga aplicatie.

Din punct de vedere al ingineriei programarii, cel mai important lucru este proiectarea arhitecturii aplicatiei. Asa cum se realizeza un plan al produsului informatic se recomanda realizarea planului interfetei cu utilizatorul prin care se clarifica cum va arata interfata grafica inainte de a trece la implementarea efectiva a acesteia.

In procesul de proiectare a interfetelor grafice se pot presupune realizarea de prototipuri ale interfetelor grafice , apoi se vor evalua si reactualiza daca este cazul pe parcursul dezvoltarii aplicatiei. Prototipurile clarifica interactiunile aplicatiei cu utilizatorii sau cu alte aplicatii. De asemenea prin acest procedeu utilizatorii sunt atrasi in procesul de realizare a aplicatiei.

Exista mai multe metode de dezvoltare a interfetelor grafice cu Java. Una din modalitatile de creare de appleturi folosind pachetul java.awt si de metodele de dezvoltare a aplicatiilor stand-alone cu pachetul java.swing;

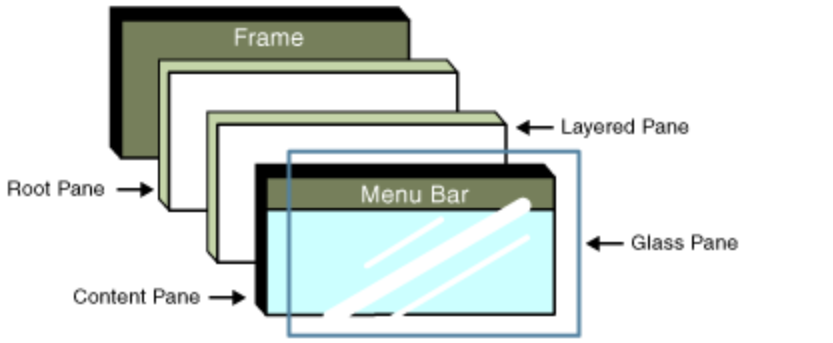
Swing este platforma java prin care se face interactiunea intre utilizator si calculator. Swing implementeaza totodata si pattern-ul de dezovoltare MVC(Model view controller) , el separa modelul interfetei de modelul aplicatiei.

Swing contine toate componentele pe care ne asteptam sa le gasim intr-o interfata moderna, este o librarie vasta si care cuprinde nivele de complexitate in functie de nevoi. Se poate modifca daca este necesar si aspectul ceea ce priveste interfata in functie de tipul de sistem de operare.

Pentru a crea o interfata grafica folosim clasa JFrame care descrie un obiect de tip fereastra. O fereastra este de asemenea un container, ceea ce inseamna ca poate sustine alte componente. Fereastra este princialul obiect de tip container folosit pentru crearea unei interfete grafice pentru o aplicatie software.

Putem observa caracteristica de container a unei ferestre din ierarhia de mostenire a claselor:

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame



From Sun Microsystems(tm) Website

(1)

Libraria Swing se poate folosi si in aplicatii dezvoltate pentru platforme Web prin intermediul unui plug-in (Java plug-in) dar si pentru appleturi Java. Datorita acestui lucru Swing inlocuieste appletul in platformele Web.

Majoritatea aplicatiilor Swing sunt construite in interiorul clasei JFrame, care creaza o fereastra in orice tip de sistem de operare folosim.

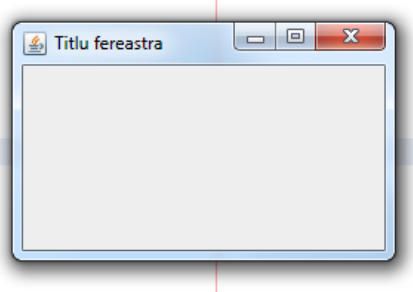
Exemplu:

```
/**
 *
 * @author mpopa
 */
public class Fereastra extends JFrame {

        public static JFrame f=new JFrame("Titlu fereastra");

        public static void main(String [] args){
            f.setVisible(true);
        }

}
```



Titlul ferestrei este dat ca parametru constructorului JFrame(), metoda setVisible(true) face fereastra vizibil. O parte din componentele librariei Swing vor fi prezentate partii practice a proiectului.

Pattern-uri de dezvoltare[3]

Un pattern reprezinta o modalitate standandard de a rezolva un tip de probleme sau de a face un anumit lucru. Pattern-ul de dezvoltare se refera la o anumita directie de dezvoltare si un anumit mod, un fel de limbaj pe care toti sa-l inteleaga cand rezolva o anumita problema sau in cazul nostru dezvoltarea unui produs software.

Scopul unui pattern este de a izola schimbarile efectuate in cod, deoarece atunci cand producem un cod vrem sa separam componentele care se pot schimba de cele care raman statice, atunci se apeleaza la abstractizare.

Pattern-urile se incadreaza in mai multe categorii:

-Pattern creational: cum un obiect poate fi creat, acest lucru de cele mai multe ori implica izolarea detalilor crearii pentru ca codul sa nu fie dependend de acest tip de obiecte.

-Pattern Structural: proiectarea unui obiect sa satisfaca anumite particularitati ale unui proiect. Acesta are legatura cu modul in care obiectele sunt cunectate intre ele pentru a asigura ca shimbarile in sistem nu ncesita schimbarea conexiunilor.

-Pattern Comportamental: obiecte care se ocupa de un anumit tip de actiuni in interiorul programului. Acestea encapsuleaza procesele pe care vrem sa le executam.

Pattern-uri creationale

Singleton

Singleton-ul este o instanta a unei clase sau unei valori accesibila global in aplicatie. Cheia crearii unui singleton este de a preveni clientul sa creeze un obiect in alt mod decat noi il declaram. Pentru a face asta trebuie sa declaram toti constructorii privati si sa setam un constructor pentru a nu lasa compilatorul sa faca unul pentru noi (constructorul default());

Prototip

Acest pattern permite clonarea obiectelor si reducerea costului de creatie. Exemplu: Suprascrierea constructorilor.

Pattern-uri Comportamentale:

Vizitator

Defineste o operatie noua pentru lucrul cu clasele din program, pentru a nu modifica structura programului.

Lantul de responsabilitati

Permite mai multor obiecte sa foloseasca o cere fara ca aceastea sa stie unul de celalalt. Transmite mai departe cererea obiectelor inaltuite pana aceasta a fost rezolvata. Exemplu: Java Servlet .

Patern-uri Java EE

Cel mai binecunoscut pattern Java EE este MVC.

MVC (Model View Controller) este un pattern arhitectural, se ocupa de relatiile intre date,reprezentare si controller. Este foarte utilizat deoarece ofera o viziune mai buna asupra codului scris, un cod mai curat, usor de parcurs, are regului clare si ofera flexibilitate mare.

Aceste modele de proiectare vor fi prezentate in capitolul urmator al proiectului, acestea fiind descrise pe parcursul prezentarii.

Prezentarea unui proces de dezvoltare proiect

In continuare, pentru a descrie mai eficient etapeele procesului de dezvoltare ale unui proiect, vom evalua un proiect implementat de echipa noastra pentru unul din laboratoare.

Acest proiect reprezinta o aplicatie Server-Client.

Etapele proiectarii au fost urmatoarele :

Etapa 1: Care sunt necesitatile clientului ?

R: Trebuie implementat un sistem de supraveghere al conturilor bancare disponibiile fiecarui client al unei banci.

Etapa 2: Cum dorim sa ii indeplinim clientului cerintele ?

R: Vom proiecta o aplicatie care lucreaza cu o un sistem centralizator de date prin care vom putea stoca, prelucra si prelua date.

Etapa 3:C are sunt necesitatile tehnologice in acest caz ?

R: Se vor folosi tehnologii de Java Swing si Controllere de XML (am ales aceste tipuri de controllere pentru ca am dorit ceva inovativ pe plan personal, noi fiind foarte familiarizati cu sistemele de controller de baze de date JDBC, JPA) care simuleaza comportamentul bazelor de date.

Etapa 4: Pe plan intern, care sunt pasii de proiectare ai aplicatiei ?

R: Arhitectura va fi bazata pe Model-View-Controller,alcatuindu-se trei ramuri principale :

Partea de Model

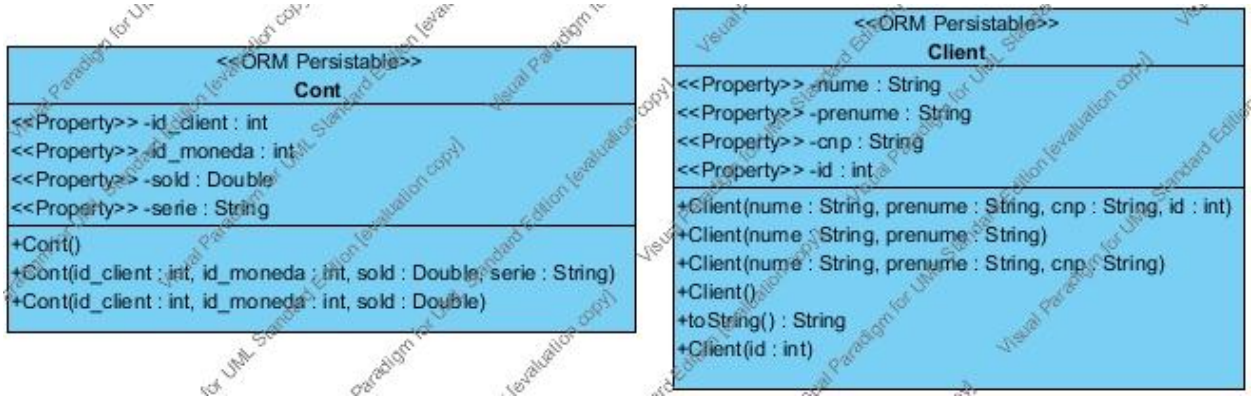
Partea de View

Partea de Controller

Partea de Model

Aici sunt inglobate datele care vor fi prelucrate. Aceasta este compusa din package-ul “beans”, unde sunt stocate clasele folosite pentru descrierea entitatilor ce vor contine datele ce sunt prelucrate de catre Server la cererea clientului. Clasele continute in acest package sunt : Client,Cont,Mesaj,Moneda,User.

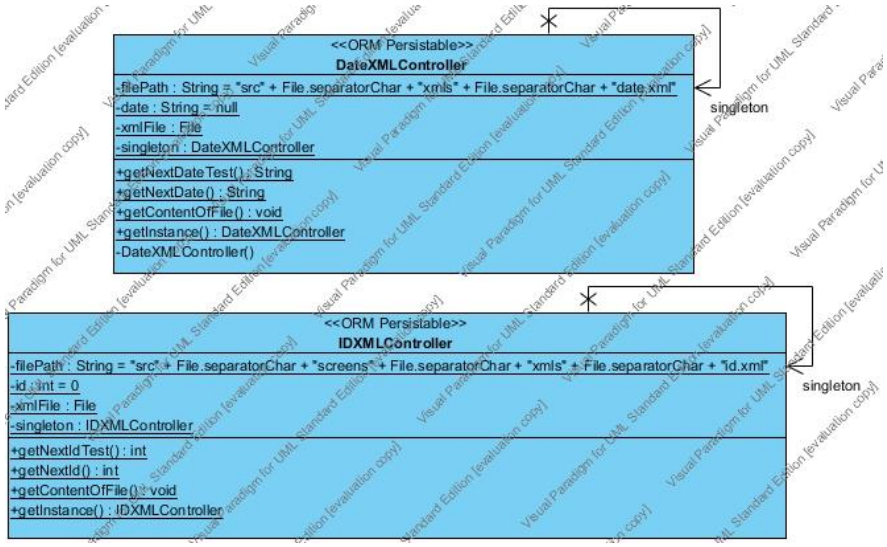
Mai jos puteti vedea diagramele UML a doua dintre clasele mentionate anterior. Dupa cum puteti vedea, acestea au campuri ce descriu coloanele unei tabele dintr-o baza de date. Aceste modele de date pot fi obtinute foarte usor prin implementarea “Create Entinty classes from Database Tables” disponibila in proiectarea JPA.



Dupa cum se poate vedea si din aceste diagrame, la proiectarea Java Beans-urilor sau POJO-urilor (Plain Old Java Object) s-au folosit principiile de incapsulare si mai ales pattern-ul de **Prototip** (ce se poate vedea din constructorii suprascrisi).

Partea de Controller

Acesta contine toate clasele specializate pentru stoca/modifica datele cuprinse in entitatile de depozitare a acestora. Controllerele sunt clase care modifica fisiere XML. Acest lucru se poate deduce si din numerele acestora : ClientiXMLController,ConturiXMLController, etc.



Aceste clase contin metodele ce vor fi expuse utilizatorului prin partea de **View**, pentru a simula cererile acestuia la sistemul creat. Dupa cum se poate observa din diagrama UML prezentata mai sus, controllerele au fost implementate folosind pattern-ul de programare **Singleton,** intrucat nu dorim ca utilizatorul sa aibe acces la constructorul acestor clase. NU dorim mai mult de o singura instanta din aceste clasa existenta pe sesiune de utilizare.

Partea de View

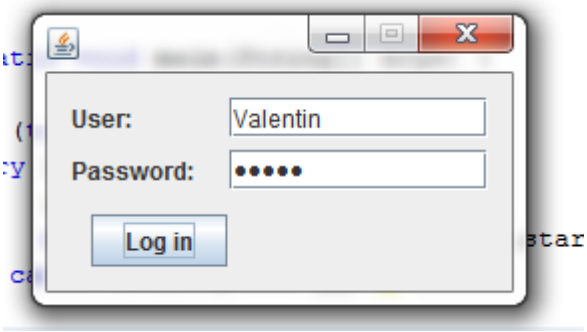
Aceasta este partea din program care ii este familiara utilizatorului, intrucat acesta lucreaza doar cu partea de front-end a proiectului. De aceea, experienta utitlizatorului trebuie sa fie de cea mai mare calitate.

Cum am mentionat si anterior, tehnologiile Swing ale limbajului Java ofera o parte vizuala foarte fluida si foarte bine pusa la punct, fara erori de imbinare a diverselor componente (erori ce sunt foarte vizibile la tehnologiile HTML/CSS, unde, daca nu se incarca o librarie, sau conexiunea la internet este foarte lenta, scripturile necesare se incarca doar partial sau deloc, ducand la un esec total al experientei utlizatorului).

Vom starui mai mult asupra acesteia, intrucat dorim sa aratam versatilitatea si complexitatea tehnologiei de User Interface.

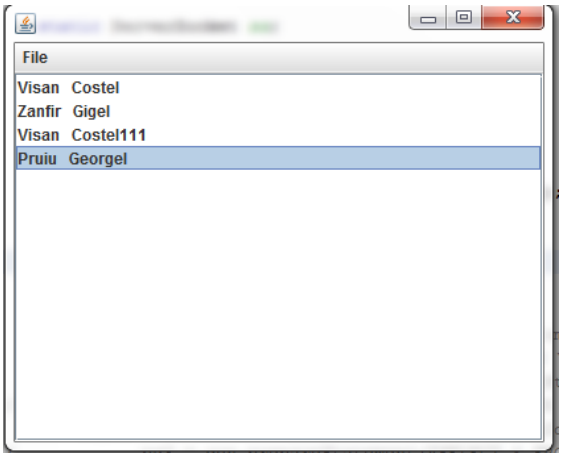
Conceptul de baza in structura acestor ecrane este unul foarte simplu. Se foloseste un container principal, care este in mod obligator o clasa care mosteneste un JFrame. Acesta va contine toate componentele cu care va interactiona utilizatorul. In unele cazuri, se mai pot folosi alte containere aditionale (JPanel) pentru a grupa mult mai eficient din punct de vedere vizual butoanele, campurile de text si alte elemente ce definesc experienta utilizatorului cu sistemul.

Trecand peste partea de Server, unde este implementat un sistem multi-thread folosind ServerSocket-uri si vectori de clase care mostenesc clasa Thread, ajungem la ecranul de logare a utilizatorului.

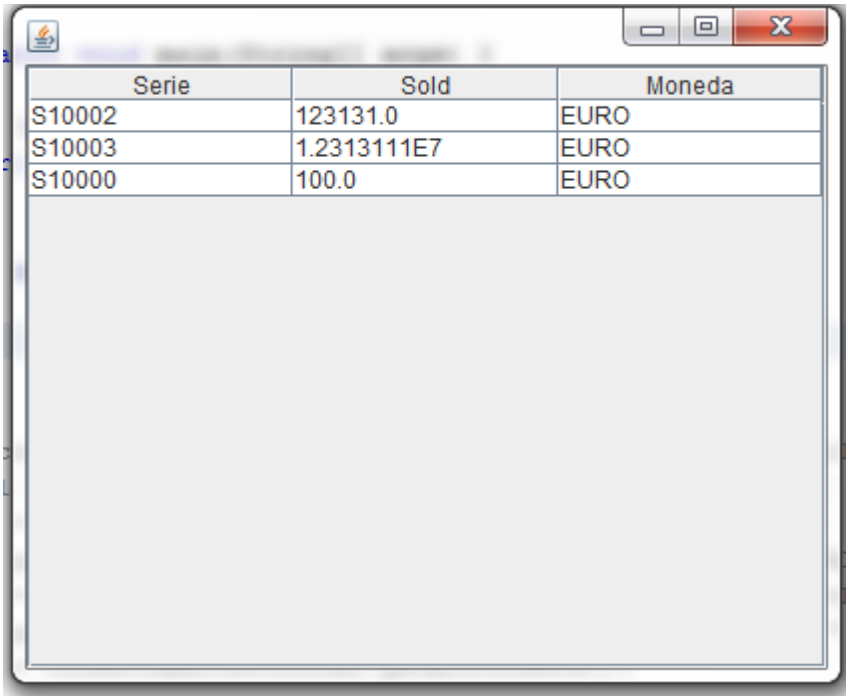


Primul ecran pe care il intalneste orice utilizator al vreunei aplicatii este cel de login. Acesta trebuie sa fie simplu si la obiect. Dupa cum se poate vedea, s-au folosit doua campuri de text (unul special pentru parola), doua etichete care sugereaza ce trebuie completat in fiecare camp si un buton.

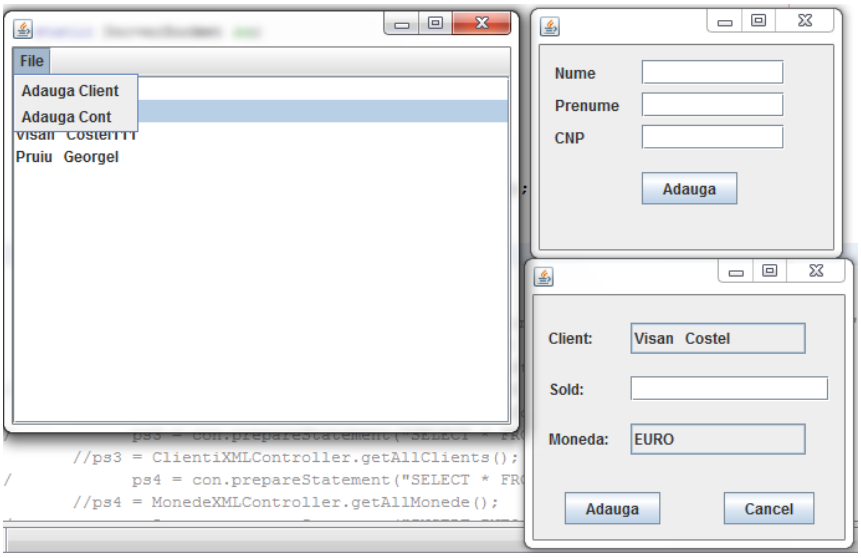
Pe buton s-a adauga un ActionListener, care asculta pentru actiuni (click) din partea utilizatorului. In momentul in care s-a dat click pe buton, in back-end, UsersXMLController, foloseste un bean al clasei User care s-a creat folosind datele din cele doua campuri de text pentru a verifica daca in baza de date curenta exista inregistrat utilizatorul cu aceste date. Daca exista, i se va permite logarea in aplicatie.



Mai departe, se ajunge intr-un ecran care contine o bara de meniu (File) cu doua optiuni (Adauga Cont si Adauga User). Sub aceasta, gasim o lista cu toti userii disponibili in sistem, dintre care multi sunt clienti activi. La dublu click pe unul dintre numele din lista



(anterior s-a selectat Pruiu Georgel) se va deschide un ecran cu lista Conturilor atasate acelui utilizator. Astfel putem vedea versatilitatea ecranelor create in Swing, intrucat constructorii clasei de JFrame permite pasarea ca parametru a unui obiect ce continea datele utilizatorului, si folosind din nou controlerii de XML au fost aduse in pagina datele atasate acelui utilizator.

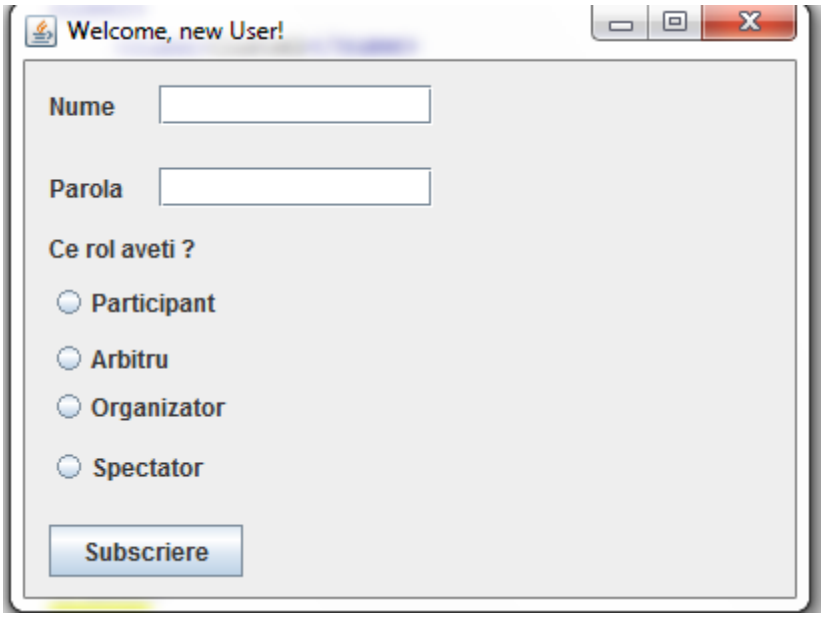


Dupa cum am mentionat anterior, meniul prezinta doua optiuni, de adaugare de client sau de cont. Afisate mai sus se gasesc cele doua ecrane, care sunt compuse din campuri text, butoane si doua ComboBox-uri, care permit utilizatorului sa aleaga valori dintr-o lista deja presetata.

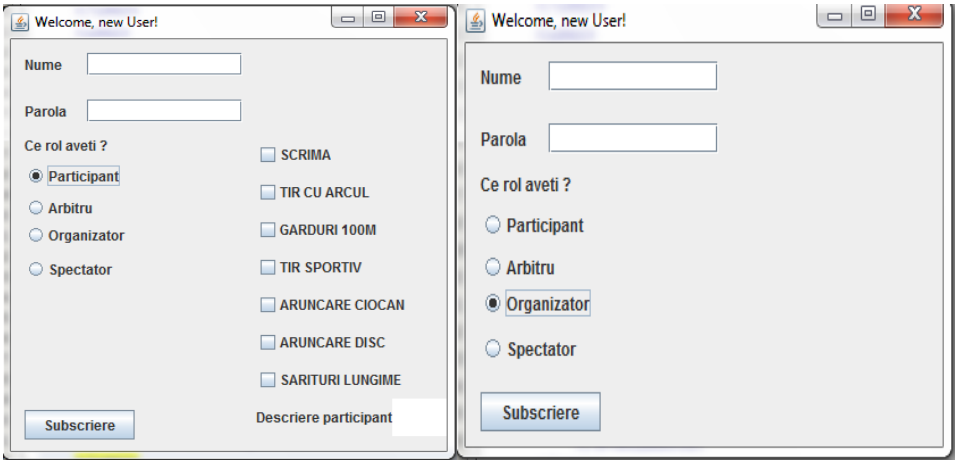
Avantajul principal al acestor ecrane este viteza cu care sunt create in timp real in functie de alegerea utilizatorului.

Un pattern foarte puternic in proiectarea de ecrane pentru utilizator este **Factory Pattern**. Insa pentru acesta, trebuie pomenita o alta aplicatie dezvoltata pentru laborator. Acesta simula desfasurarea unei competitii sportive. La login, daca utilizatorul era unul nou, acestea avea optiunea de inscriere (de mentionat ca in aplicatia prezentata anterior, nu exista aceasta optiune).

In acel punct, utilizatorul era prezentat cu urmatorul ecran :



Ce este special la acest ecran ? Faptul ca in functie de ce selecteaza utilizatorul dintre cele 4 RadioButton-uri prezente pe ecran, in partea dreapta se creeaza un panou special doar pentru acel tip de rol.



Aici este prezentat avantajul folosirii **Factory Pattern**-ului. S-a adaugat posibilitatea selectarii mai multor evenimente pentru competitor la care sa participe direct din ecranul de inscriere. Astfel s-a eliminat nevoia crearii unui nou ecran necesar pentru modificarea aditionala a datelor.

Dupa cum puteti vedea in diverse formulare de inscriere, se foloseste modelul de **Bread Crumbs**, adica dupa ce sunt completate datele de pe primul formular (date generale : Nume, prenume), s-ar fi trecut la al doilea ecran, pentru selectarea concursurilor la care sa participe. Prin **Factory Pattern** implementat direct in primul ecran, s-a eliminat direct cel putin un ecran care ar fi facut experienta utilizatorului mai putin placuta, cel putin pe acest plan.

ORACLE ADF [4]

Platforma Java EE

Java EE defineste o platform pentru dezvoltarea, instalarea si executia aplicatiilor intr-un model de aplicatie distribuit, cu mai multe ramuri. Adica, logica unei aplicatii Java EE poate fi sparta in mai multe componente bazate pe functiile acestora si distribuite pe ramura corespunzatoare a arhitecturii multi-ramificata. De exemplu, aceste componente sunt :

1. Componente de rang Client (un web browser) sunt rulate pe masina clientului.'
2. Logica de prezentare este construita cu componente de tip Web, cum ar fi JavaServer Faces (JSF) si servlet-uri Java care ruleaza pe serverul Java EE.
3. Logica de business de partea serverului este distribuita ca componente de ramura-business, care ruleaza pe serverul de Java EE. Exemple de astfel de componente sunt EJB-uri (Enterprise Java Beans) si POJO-uri.
4. Data este stocata in EIS (Enterprise Information Systems) care ruleaza pe serverul baza de date, cum ar fi Oracle Database 11g.

Pe noi ne intereseaza in principal punctul 2, componentele de tip web. Compontele Java EE de tip Web-tier pot fi servleti, sau JSP-uri care pot genera HTML, WML (Wireless Markup Language) sau XML (Extensible Markup Language) in mod dinamic sau static. Servletii Java ofera un API simplu dar puternic pentru generare de pagini Web prin procesarea in mod dinamic a cererii clientilor si construirea raspunsurilor. JSP-urile simplifica procesul de generare dinamica a continutului prin permiterea folosirii a limbajului Java ca limbaj de scriptare in interiorul paginilor HTML. JSP-urile sunt translatate in Servleti Java, care sunt apoi rulati pe un server Web ca orice alt Servlet.

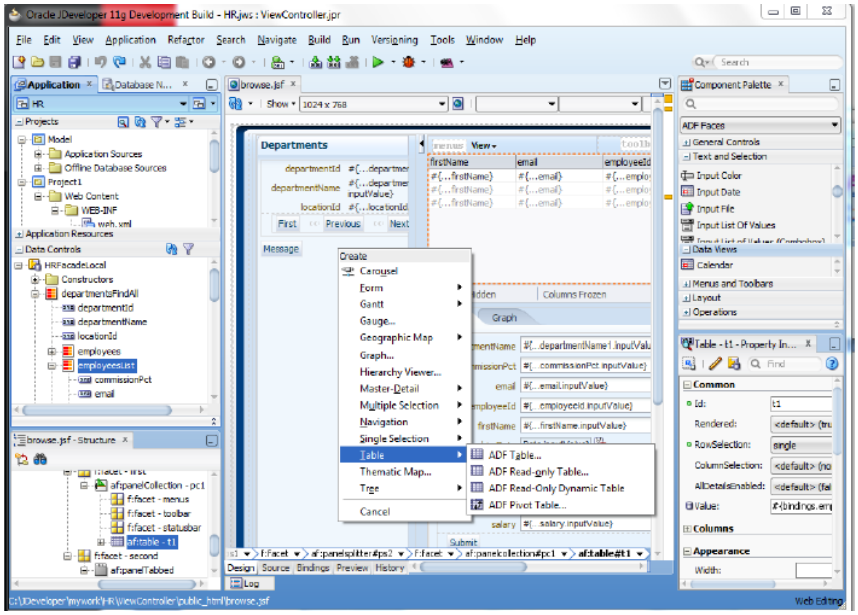
Componentele Web pot accesa componentele de business care acceseaza baza de date prin folosirea JDBC. Componentele Web pot prelucra cereri de la client cum ar fi submisii de la formulare. In continuare sunt prezentate cateva avantaje ale folosirii componentelor Web:

1. Interfata HTML care este generate de o componenta web este usoara in comparative cu un applet care necesita download-uri masive din partea clientului la rulare.
2. Interfata HTML nu necesita o instalare initiala pe masinile client, in timp ce clientii conventionali necesita reinstalarea aplicatiei pe masinile client.
3. Protocolul HTML, prin care este facuta cererea pentru pagina Web de catre client, poate trece prin mai toate firewall-urile.

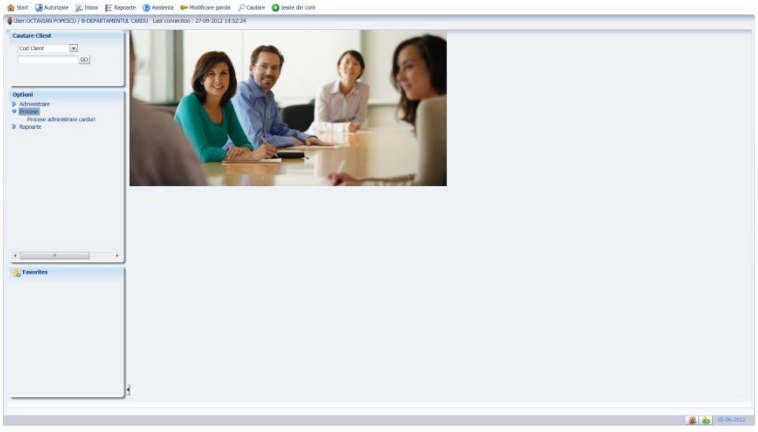
De ce ADF ?[5]

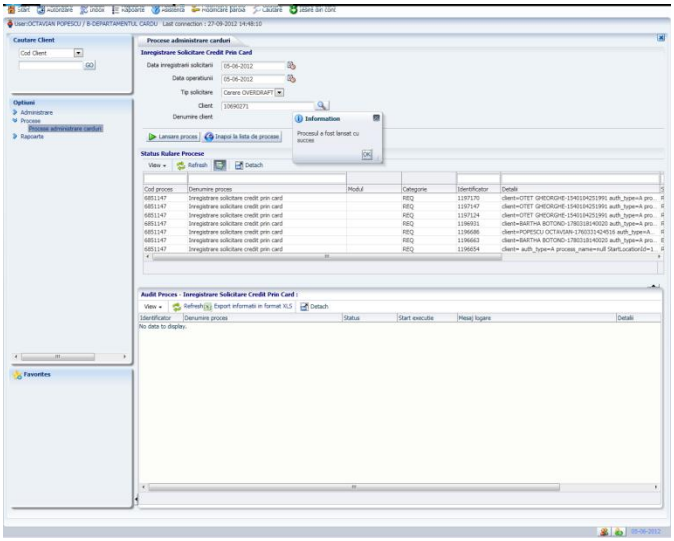
Oracle Application Development Framework (Oracle ADF) este un framework inovativ,dar matur, de dezvoltare Java EE. Oracle ADF simplifica dezvoltarea Java EE prin minimizarea nevoii de scriere de cod care implementeaza infrastructura aplicatiei permitand dezvoltatorilor sa se concentreze asupra detaliilor ce privesc aplicatia in sine.

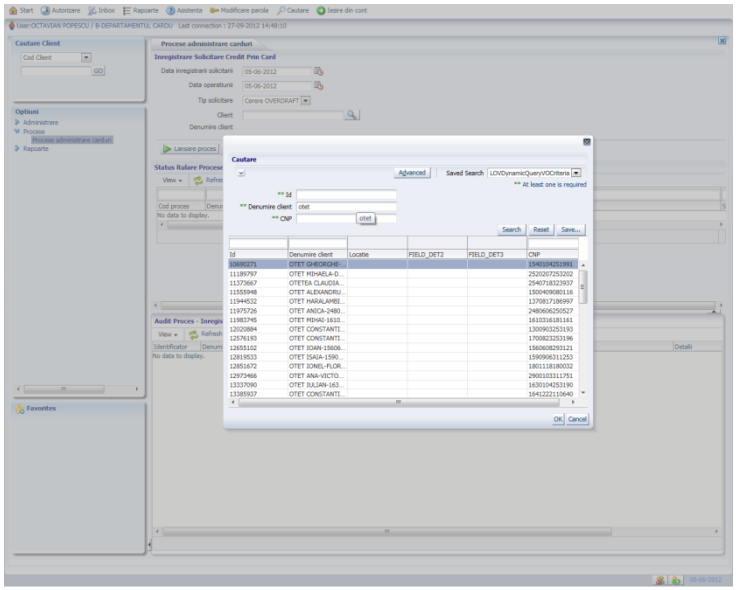
Oracle ADF implementeaza pattern-ul MVC si ofera solutii care acopera toate zonele de dezvoltare, cum ar fi : maparea Obiect/Relatie, persistenta datelor, strat de controller reutilizabil, framework de interfata Web bogat, legarea (binding) datelor de interfata de utilizator si multe altele.



Prin implementarea unui mediu de dezvoltare inovator (JDeveloper) care permite folosirea legaturilor de date doar prin “drag-and-drop”, munca de proiectare este simplificata,







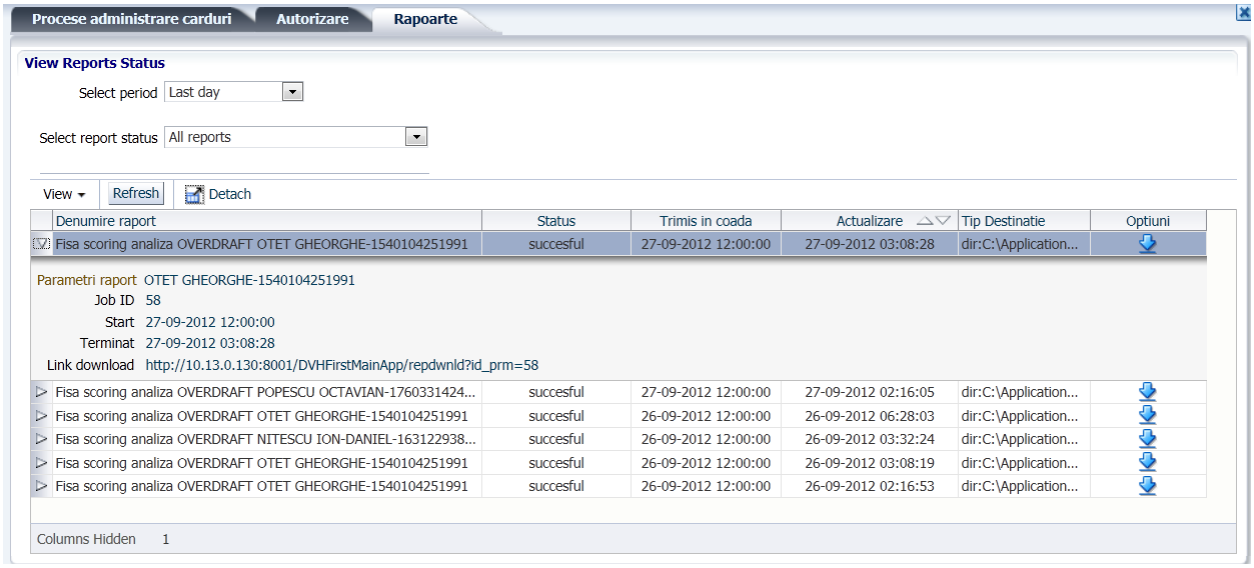
iar interfetele utilizatorilor sunt simple dar pot contine foarte multe informatii.

Desi ADF poate fi considerat doar o continuare a deja existentelor framework-uri de Web development, inovatia adusa de acesta consta in paginile **.jsff**. Acestea sunt “java server faces fragments”, adica pagini JSF sub forma de pagini xml.

Acest lucru permite programatorului sa poata incarca intr-o singura pagina .jspx sau .jsf, mai multe pagini .jsff. Astfel, managementul paginilor este usurat, intrucat acestea nu mai sunt conglomerate de cod html de sute de randuri, ci bucati mici bine structurate.

Cele trei poze prezentate mai sus de fapt prezinta aceeasi pagina, index.jspx care este compusa initial din bara de meniu superioara si bara din stanga. Acesteia, in functie de ce selecteaza utilizatorul, i se adauga alte pagini in zona centrala dand impresia unei singure pagini foarte versatile. Dar de fapt, controlerul din spate aduce, de fiecare data, o pagina noua in functie de cerinta celui care interactioneaza cu aplicatia.

De asemenea, faptul ca aceste pagini pot fi incarcate intr-un numar considerabil, permite organizarea acestora sub forma de tab-uri in ecranul utilizatorului (dupa cum se vede in imaginea de mai jos), astfel facand tranzitia intre pagini mult mai facila si mult mai simpla pentru end-user.



Concluzii

Desi tehnologiile aduse de framework-urile web sunt foarte ispititoare si sunt mult mai eficiente din punct de vedere al experientei grafice pentru utilizator, nu trebuie ignorata utilitatea vitezei cu care pot fi proiectate interfetele Swing si de viteza cu care acestea raspund cerintei utilizatorului in timp real.

Intr-adevar, in procesul dezvoltarii de aplicatii ADF s-au observant unele probleme cu latenta programelor, intrucat consuma multe resurse (mai ales serverele weblogic pe care sunt rulate aplicatiile), mai ales la statii vechi, insa faptul ca se pot proiecta legaturi puternice intre datele din baza de date si ecran si faptul ca pattern-ul de proiectare poate fi poreclit “pagini in pagini”, aduc avantajul acela fin care il reprezinta framework-ul ADF fata de celelalte framework-uri, mai ales fata de Swing.

Bibliografie

[1]Andrew Tanenbaum - Software Engineering

[2]Bruce Eckel - Thinking in Java, 4thEdition

[3]Bruce Eckel - Thinking in Patterns with Java

[4]Fusion Middleware 11g Building ADF Faces Clients for EJB and JPA

[5]Oracle Fusion Middleware 11g Build Applications with ADF

Responsabilitatile fiecarui student :

- Colesnicencu Claudiu
    - Oracle ADF
    - De ce ADF?
    - Concluzii
- Popa Marian-Gabriel
    - Introducere in Java si GUI
    - Pattern-uri de dezvoltare
- Visan Valentin
    - Prezentarea unui proces de dezvoltare proiect
    - Dezvoltare aplicatii folosite pentru exemplificarea pattern-urilor