Tehnici de programare. Note de laborator

Danciu Gabriel Mihail

Universitatea Transilvania, Facultatea de Inginerie Electrică și Știința Calculatoarelor

Cuprins

1 Laboratorul1. Introducere in Java Swing

1.1 Versionarea implmentării

Toate resursele laboratoarelor se vor afla pe github la adresa: https://github.com/danciugaby/TehniciProgramare2016. Necesitatea versionării codului unei aplicații apare din mai multe motive. Să considerăm că doar o persoană a participat la scrierea aplicației. Dacă la un moment dat alterează o funcționalitate și salvează modificarea iar codul nu mai compilează sau nu se mai execută corect atunci nu există posibilitatea de revenire la un stadiu anterior. Dacă la aplicație lucrează mai multe persoane, este nevoie de combinarea funcționalităților implmementate. Mai multe argumente în favoarea folosirii versionării codului se găsesc aici.

Urmărirea problemelor apărute pe măsură ce implementarea avansează așa numitul "bug tracking" se poate realiza cu aplicații deja existente Bugzilla sau Jira. De asemenea ele pot fi integrate în uneltele de dezvoltare precum TFS sau Git.

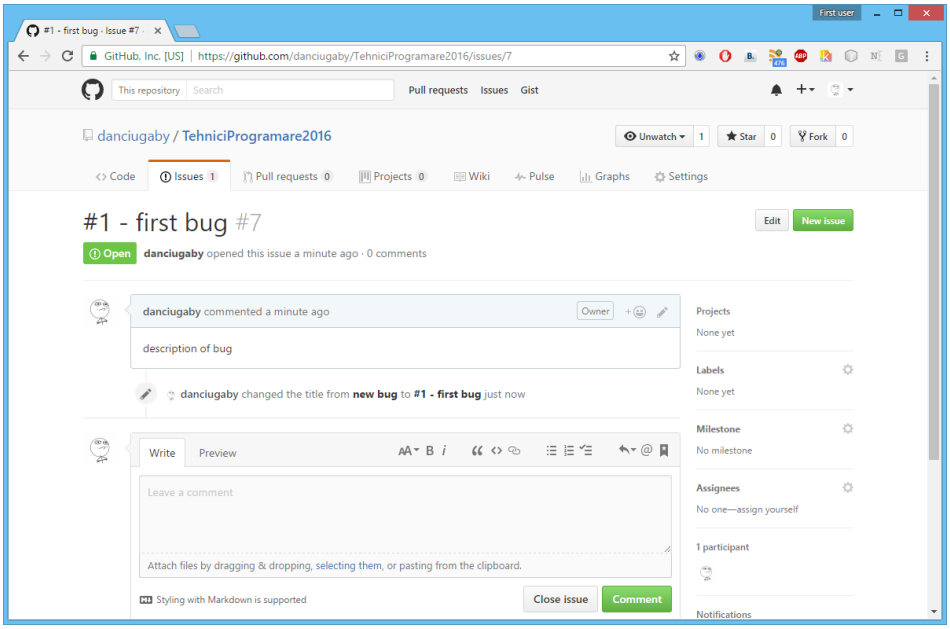

Figura 1. Exemplu de bug tracking

1.2 Aspecte generale

Swing API este un set de componente de intergafa grafica ce ajuta dezvoltatorul sa creeze aplicatii pentru interactiunea cu utilizatorul. Swing are la baza API-ul AWT si functioneaza ca un inlocuitor al acestuia, continand aproape toate controalele AWT-ului.

In cele ce urmeaza vom folosi termenul Componenta pentru a referi un set de controale grafice dispuse pe interfata, cat si datele ce sunt prelucrate prin interactiunea utilizatorului cu alutorul acestor controale.

Swing se bazeaza pe arhitectura MVC (Model-View-Controler), model ce presupune urmatoarele concepte:

1. Un Model contine datele folosite de o componenta.
2. View-ul contine reprezentarea vizuala a componentei.
3. Controler-ul preia input-ul de la utilizator si transmite modificarile catre datele componentei.

Model-ul va fi mereu separat de partea vizuala ce va contine View-ul si Controler-ul.

1.3 Notiuni introductive

In primele laboratoare se dorește acomodarea cu tehnologia Java pentru crearea de interfețe grafice. Vom incerca sa acoperi parțial tutorialul de la pagina: https://docs.oracle.com/javase/tutorial/uiswing/TOC.html.

Elemente UI - Sunt elementele visuale ce sunt afisate userului si cu care acesta interactioneaza.

Layouts - Definesc organizarea pe ecran a elementelor vizuale si asigura o prezentare unitara pentru GUI (Graphic User Interface).

Behavior - Reprezinta setul de evenimente ce au loc atunci cand utilizatorul interactioneaza cu interfata grafica.

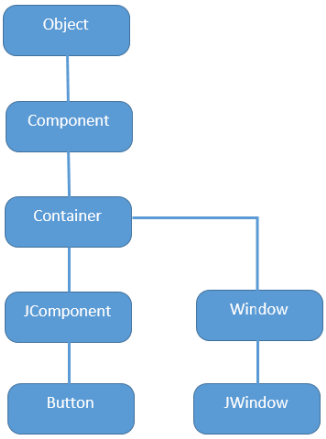

Figura 2. Ierarhia claselor din pachetul Swing

Component - Este o clasa de baza abstracta pentru controalele UI (exceptand meniurile) ale SWING. Pe scurt, putem spune ca reprezinta un obiect ce are o reprezentare grafica.

Container - Este o componenta ce poate contine alte componenete SWING.

JComponent - Este o clasa de baza pentru toate componenetele UI ale SWING. Pentru a folosi o componeneta swing ce mosteneste clasa JComponent, aceasta trebuie sa fie continuta intr-o ierarhie a carei radacina sa fie un container swing de nivel inalt.

1.4 Elemente UI Swing

JLabel - O componenta folosita pentru adaugarea unui text intr-un container.

JButton - O clasa pentru crearea unui buton cu eticheta.

JCheckBox - O componenta grafica ce poate fi setata pe starea activa (true) sau inactiva (false).

JRadioButton - O componeneta grafica activa/inactiva ce face parte dintr-un grup.

JList - O componeneta ce ofera utilizatorului o lista de obiecte text.

JComboBox - O componenta ce ofera utilizatorul posibilitatea de a alege dintr-un meniu de optiuni.

JTextField - O componenta UI ce ofera utilizatorului posibilitatea de a edita o singura line de text.

JTextArea - O componenta UI ce ofera utilizatorului posibilitatea de a edita mai multe linii de text.

JPasswordField - O componenta text specializata pentru introducerea de parole.

JColorChooser - Ofera o serie de cotroale ce permite utilizatorului sa selecteze si sa foloseasca o culoare.

ImageIcon - O implementare a interfetei Icon ce deseneaza iconite din imagini.

JScrollbar - O componeneta ce permite navigarea intr-o lista de item-uri.

JOptionPane - Un set de controale de tip dialog standard folosite pentru interactiunea cu utilizatorul.

JFileChooser - Reprezinta o fereastra de tip dialog ce permite utilizatorului sa selecteze un fisier.

JProgressBar - O componeneta ce afiseaza progresul unui task ce ruleaza la un moment dat.

JSlider - O componeneta grafica ce permite selectia dintr-un interval.

JSpinner - Este un camp cu o singura linie ce permite utilizatorului sa selecteze un numar sau valoarea unui obiect dintr-o secventa ordonata.

1.5 Containere

Swing pune la dispozitie trei clase container de nivel avansat: JFrame, JDialog si JApplet. Este bine de retinut in utilizarea acestor clase:

- Pentru a aparea pe ecran, orice componenta GUI trebuie sa faca parte dintr-o ierarhie a unui container. Aceasta ierarhie reprezinta este un arbore de componente ce are ca parinte un container de nivel inalt.
- Orice componenta GUI poate fi continuta intr-un singur container. Daca o componenta apartine deja unui container si se adauga unui alt container, aceasta va fi scoasa din primul container si adaugata in al doilea.
- Fiecare container de nivel inalt are o zona de continut, ce contine (direct sau indirect) componentele vizibile din acel container.
- Optional, se poate adauga o bara de meniu unui container de nivel inalt. Aceasta este pozitionata in container, dar in afara zonei cu continut.

De retinut: Cu toate ca JInternalFrames este foarte asemanator cu JFrame, aceste containere nu sunt de nivel inalt.

Mai jos este un exemplu de o fereastra creata intr-o aplicatie. Aceasta contine o bara de meniuri goala, iar in zona cu continut un label simplu, galben: http://docs.oracle.com/javase/tutorial/figures/uiswing/components/TopLevelDemoMetal.png.Aceasta este ierarhia de continut pentru acest simplu exemplu de GUI: http://docs.oracle.com/javase/tutorial/figures/uiswing/components/3jframe.gif

Containerele de nivel inalt si Ierarhiile de continut Fiecare program ce foloseste componente Swing are cel putin un container de nivel inalt. Acesta este radacina unei ierarhii de continut - ierarhia ce contine toate componentele Swing ce apar in containerul de nivel inalt.

Ca o regula, o aplicatie de sine statatoare cu un GUI bazat pe Swing are cel putin o ierarhie ce continut cu un JFrame ca radacina. De exemplu, daca o aplicatie are o fereastra principala si doua dialoguri, atunci aceasta are trei ierarhii de continut, si deci trei containere de nivel inalt. Ierarhia de continut pentru fereastra principala va avea un JFrame drept container, iar ierarhiile asociate dialogurilor vor avea drept containere radacina obiecte de tip JDialog.

Un applet bazat pe Swing are cel putin o ierarhie de continut, dintre care, exact una are ca radacina un obiect JApplet. De exemplu, un applet ce deschide un dialog va avea doua ierarhii de continut. Componentele din fereastra browser vor fi plasate intr-o ierarhie de continut ce va avea ca radacina un obiect JApplet. Dialogul in schimb va avea ca radacina pentru ierarhia de continut un obiect JDialog.

Adaugarea de componente la un panou cu continute Revenind la exemplul de mai sus, acesta este codul folosit pentru a obtine zona(panou) cu continut a unui frame si a-i adauga label-ul galben:

```
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
```

Dupa cum se vede, getContentPane este metoda ce returneaza panoul de continut a unui container de nivel inalt. Zona principala default o reprezinta un simplu container intermediar ce mosteneste din JComponent, si care foloseste BorderLayout drept layout manager.

Un panou cu continut poate fi usor customizabil, de exemplu prin atribuirea unui layout manager sau prin adaugarea unei margini. De retinut insa ca getContentPane returneaza un obiect Container, si nu JComponent. Asta inseamna ca pentru a putea folosi atributele JComponent, trebuie fie sa se faca conversia obiectului returnat, fie sa se creeze o componenta proprie ce va fi folosita drept panoul cu continut. In cazurile pe care le vom trata in continuare vom folosi cea de-a doua solutie.

Pentru crea o componenta de tip panou cu continut, se foloseste metoda setContentPane a container-ului radacina. De exemplu:

```
// Creaza un panou si adauga componente la ea
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);
topLevelContainer.setContentPane(contentPane);
```

De retinut:

Pentru usurinta, metodele de adaugare, stergere si atribuire a unui layout au fost suprascrise pentru a trimite actiunea catre panoul principal daca e necesar. Asta inseamna ca se poate scrie:

```
frame.add(child);
```

iar obiectul ”child” va fi adaugat zonei principale. DOAR aceste trei metode au acest comportament. Asta inseamna ca metoda getLayout() nu va returna layout-ul atribuit prin metoda setLayout().

Adaugarea barei de meniu Teoretic, toate container-ele de nivel inalt pot contine o bara de meniu. In practica insa, barele de meniu apar doar in frame-uri si applet-uri. Pentru a adauga o bara de meniuri la un container de nivel inalt, se creaza un obiect de tipul JMenuBar, se populeaza cu meniuri, si apoi se apeleaza din container setMenuBar.

```
frame.setJMenuBar(greenMenuBar);
```

Fiecare container de nivel inalt se bazeaza pe un container numit panoul radacina. Panoul radacina se ocupa de panoul principal si de bara de meniuri, impreuna cu un set de alte containere. Este nevoie de o cunoastere mai buna a modului in care lucreaza panoul radacina in momentul in care este nevoie sa se intercepteze click-urile mouse-ului sau sa se deseneze in mai multe componenete.

In imaginea alaturata vedem lista de componente pusa la dispozitie de panoul radacina oricarui container de nivel inalt: un panou stratificat, o bara de meniuri, o zona principala si o zona transparenta. http://docs.oracle.com/javase/tutorial/figures/ui/ui-rootPane.gif

Panoul stratificat contine bara de meniuri si panoul principal si face posibila ordonarea pe axa Z a celorlalte componente. Zona transparenta este de obicei folosita pentru a prinde evenimente input ce au loc in containerul de nivel inalt, dar poate fi folosit si pentru desenarea peste mai multe componente.

1.6 Temă

Sa se parcurga toate exemplele din Lab1. Sa se implementeze o aplicație ce conține o singura forma cu 2 controale ce permit introducerea unor numere. Pe aceeasi forma se va gasi un control ce va permite alegerea unei operatiuni matematice: ”* / + -” ce se va aplica asupra celor 2 numere. Un buton va declansa acest calcul ce va avea ca raspuns afisarea pe un alt control a rezultatului operatiunii matematice.

2 Laboratorul 2. Componente GUI partea a doua.

2.1 Panouri stratificate

Un panou stratificat este un container Swing ce pune la dispozitie o a treia dimensiune pentru pozitionarea componentelor: adancimea, cunoscuta si ca axa Z. Cand se adauga o componenta la un panou stratificat, adancimea (trebuie?) se specifica ca un numar intreg. Cu cat acesta este mai mare, cu atat componenta este mai aproape de "suprafata" container-ului. In cazul componentelor ce se suprapun, componentele mai aproape de suprafata vor fi desenate peste componentele aflate "mai jos". In cazul componentelor aflate de acelasi nivel de adancime, decizia de desenare se va lua in functie de pozitia componentelor pe nivel.

De retinut: Container-ul AWT are un API ce permite manipularea comonentelor in functie de ordinea pe Z.

Fiecare container Swing ce are un panou radacina, cum ar fi JFrame, JApplet, JDialog sau JInternalFrame, are automat si un panou stratificat, dar majoritatea aplicatiilor nu folosesc in mod explicit panoul stratificat asociat panoului radacina.

Swing pune la dispozitie doua clase pentru panouri stratificate:

- JLayeredPane - clasa folosita de panourile radacina. Aceasta clasa va fi folosita in exemplele ce urmeaza.
- JDesktopPane - este o subclasa a clasei JLayeredPane, ce este specializata in a tine frame-uri interne.

Un exemplu de aplicatie ce creaza un panou stratificat si adauga etichete colorate suprapuse pe diferite nivele de adancime: https://docs.oracle.com/javase/tutorial/uiswing/components/layeredpane.html

Adaugarea de componente la panoul stratificat se face folosind o metoda specifica. Cand o componenta este adaugata, se specifica adancimea, si optional, pozitia in cadrul nivelului de adancime. Ulterior, adancimea si pozitia pe nivelul de adancime al componentei vor putea fi modificate dinamic.

Mai jos este codul ce adauga etichetele colorate la panoul stratificat din exemplul anterior:

```
for (int i = 0; i < ...number of labels...; i++) {
        JLabel label = createColoredLabel(...);
        layeredPane.add(label, new Integer(i));
        ...
}
```

Metoda createColoredLabel(...) creaza un obiect JLabel opac si ii atribuie o culoare de fundal, o margine, un text si dimensiuni. Metoda de adaugarea folosita este cea cu doua argumente, specificandu-se ca prim argument componenta de adaugat si ca al doilea argument adancimea sa. Valorile actuale ale adancimilor componentelor nu conteaza, ci doar valorile lor relative si o folosire a lor adecvata in cadrul programului.

Pozitia unei componente in cadrul unui nivel de adancime poate fi modificata in mod dinamic. In exemplul folosit anterior, controlul CheckBox este folosit pentru a modifica pozitia etichetei Duke in cadrul nivelului de adancime. Actiunea executata pentru a efectua aceasta este:

```
public void actionPerformed(ActionEvent e) {
        if (onTop.isSelected())
        layeredPane.moveToFront(dukeLabel);
        else
        layeredPane.moveToBack(dukeLabel);
}
```

In mod default un panou stratificat nu are un manager de layout. Aceasta inseamna ca programatorul trebuie sa scrie codul necesar pentru pozitionarea si dimensionarea componentelor adaugate unui panou stratificat.

2.2 Tab-uri

Clasa JTabbedPane permite folosirea aceluiasi spatiu de mai multe componente.Utilizatorul alege ce componenta vizualizeaza prin selectarea tab-ului corespunzator componentei dorite.

Mai intai se instantiaza clasa JTabbedPane, se creaza componentele ce se doresc a fi afisate si apoi acestea se adauga la panoul Tab folosind metoda "addTab". Urmatoarea imagine este luata din aplicatia TabbedPaneDemo ce contine un panou Tab cu patru tab-uri https://docs.oracle.com/javase/tutorial/figures/uiswing/components/TabbedPaneDemo.png

In mod obisnuit, amplasarea Tab-urilor se face in partea de sus asa cum se vede in imagine. Se poate insa modifica intre stanga, dreapta, sus si jos prin folosirea metodei setTabPlacement.

Urmatoarea secventa de cod este luata din aplicatia demo TabbedPaneDemo si creaza panoul cu Tab-uri din imaginea de mai sus. De retinut ca nu este nevoie de implementarea unor metode pentru tratarea evenimentelor. Obiectul JTabbedPane are grija de evenimentele de mouse si tastatura ascoiate.

```
JTabbedPane tabbedPane = new JTabbedPane();
ImageIcon icon = createImageIcon("images/middle.gif");

JComponent panel1 = makeTextPanel("Panel #1");
tabbedPane.addTab("Tab 1", icon, panel1,
"Does nothing");
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

JComponent panel2 = makeTextPanel("Panel #2");
tabbedPane.addTab("Tab 2", icon, panel2,
"Does twice as much nothing");
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

JComponent panel3 = makeTextPanel("Panel #3");
tabbedPane.addTab("Tab 3", icon, panel3,
"Still does nothing");
tabbedPane.setMnemonicAt(2, KeyEvent.VK_3);

JComponent panel4 = makeTextPanel(
"Panel #4 (has a preferred size of 410 x 50).");
panel4.setPreferredSize(new Dimension(410, 50));
tabbedPane.addTab("Tab 4", icon, panel4,
"Does nothing at all");
tabbedPane.setMnemonicAt(3, KeyEvent.VK_4);
```

Asa cum se observa, metoda addTab este cea care se ocupa de asignarea componentelor in Tab-urile panoului. Metoda addTab are mai multe supraincarcari, dar toate au in comun ca parametri un titlu pentru Tab-ul creat si componenta ce va fi afisata. Optional se poate specifica o iconita si un text pentru tooltip. Textul si iconita pot fi ambele valori null.

O alta metoda ce poate fi folosita pentru a crea un Tab este insertTab, ce permite speficicarea index-ului asociat tab-ului creat. De retinut ca metoda addTab nu permite specificarea unui index la acest pas.

Cand se creaza componentele ce urmeaza a fi adaugate la un panou de Tab-uri, trebuie retinut ca indiferent de care tab este vizibil, fiecare copil are la dispozitie acelasi spatiu pentru a fi afisat. Dimensiunea dorita a unui panou de tab-uri este suficient de mare pentru a putea afisa complet copilul cu cea mai mare inaltime cat si copilul cu cea mai mare latime. Deasemenea, dimensiunea minima a unui panou de tab-uri depinde de cele mai mari valori minime ale inaltimilor si latimilor corespunzatoare tuturor copiilor sai.

In exemplul anterior, TabbedPaneDemo, a patra componenta are dimensiunile dorite mai mari decat celelalte componenete. Astfel, dimensiunile dorite ale panoului de Tab-uri sunt cele suficiente pentru a afisa corect componenta din tab-ul al patrulea. Toate componentele vor primi acelasi spatiu de afisare, in cazul de fata, spatiul necesar afisarii componentei patru.

Exista trei modalitati de a naviga la un aumit tab folosind GUI-ul:

1. Folosind mouse-ul.
2. Folosind sagetile de la tastatura cand obiectul JTabbedPane are focusul.
3. Folosind combinatii de taste. Metoda setMnemonicAt permite utilizatorului sa navigheze intre Tab-uri folosind anumite combinatii de taste. Spre exemplu, apelul setMnemonicAt(3, KeyEvent.VK4) asigneaza tasta 4 tab-ului al patrulea (ce are index-ul 3 datorita indexarii de la 0). Astfel, apasand Alt si 4 duce la vizualizarea componentei din al patrulea tab. De cele mai multe ori combinatia de taste de face folosind un caracter ce apare in titlul Tab-ului, acesta fiind automat marcat vizual.

2.3 ToolBar-uri

Un obiect JToolBar este un container ce grupeaza anumite componente (de obicei butoane si iconite) intr-o linie sau o coloana. De obicei, barele de instrumente permit accesul rapid la functionalitati ce sunt incluse si in meniuri.

Imaginea de mai jos arata o aplicatie numita ToolBarDemo ce contine o bara de instrumente situata deasupra zonei text https://docs.oracle.com/javase/tutorial/figures/uiswing/components/ToolBarDemo.png

In modul default, utilizatorul poate trage si muta bara de instrumente pe alta margine a container-ului sau in afara container-ului intr-o fereastra proprie. In urmatoarea imagine bara de instrumente a fost mutata pe marginea din dreapta a container-ului https://docs.oracle.com/javase/tutorial/figures/uiswing/components/ToolBarDemo-2.png

Pentru ca actiunea de mutare sa functioneze corect, bara de instrumente trebuie sa fie intr-un container ce foloseste managerul de layout: BorderLayout. Componenta influentata de bara de instrumente se afla de obicei in centrul container-ului. Bara de instrumente trebuie sa fie singura componenta aflata in plus in container si nu trebuie sa fie in centru.

Urmatoarea imagine arata aplicatia dupa ce utilizatorul a tras bara de instrumente in afara container-ului, intr-o fereastra proprie: https://docs.oracle.com/javase/tutorial/figures/uiswing/components/ToolBarDemo-3.png

Urmatoarea secventa de cod creaza o bara de instrumente si o adauga la un container (exemplu din ToolBarDemo.java)

```
public class ToolBarDemo extends JPanel
implements ActionListener {
        ...
        public ToolBarDemo() {
                super(new BorderLayout());
                ...
                JToolBar toolBar = new JToolBar("Still draggable");
                addButtons(toolBar);
                ...
                setPreferredSize(new Dimension(450, 130));
                add(toolBar, BorderLayout.PAGE_START);
                add(scrollPane, BorderLayout.CENTER);
        }
        ...
}
```

Acest cod pozitioneaza bara de instrumente deasupra panoului ”scroolPane” prin plasarea ambelor componente intr-o zona (panel) controlata de un manager BorderLayout. Bara de meniuri este pe pozitia PAGE_START si panoul de baza este pe pozitia CENTER. Deoarece aceste doua componente sunt singurele in container, iar bara de instrumente este pozitionata pe margine, aceasta poate fi trasa pe oricare alta margine a container-ului sau in afara lui, caz in care fereastra proprie a barei de instrumente va avea titlul mentionat in constructor: ”Still draggable”.

2.4 Meniuri

Un meniu este o modalitate eleganta de a permite utilizatorului sa aleaga dintr-o serie de optiuni. Exista si alte componente ce fac aceasta, cum ar fi combobox-uri, liste, butoane radio etc.

Meniurile sunt independete de celelalte componente UI. Aceasta componenta este integrată ca o bară de meniuri sau un meniu popup. O bara de meniuri conține unul sau mai multe meniuri situate de obicei în partea de sus a ferestrei principale. Un meniu popup este strans legat de pozitia cursorului.

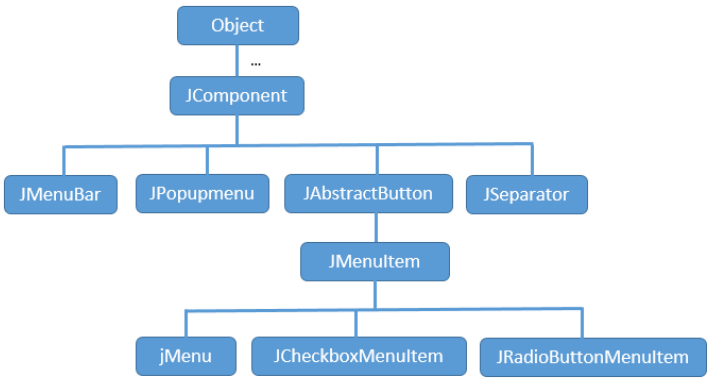

Figura 3. Ierarhia claselor meniu din pachetul Swing

Un exemplu de creare si folosire a meniurilor se afla la https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

Meniurile suporta doua tipuri de interacțiune cu tastatura: mnemotehnic și acceleratoare. Mnemonicele ofera un mod de a folosi tastatura pentru a naviga în cadrul ierarhiei meniului. Acceleratoarele, pe de altă parte, oferă o modalitate de a ocoli navigarea prin ierarhia meniului.

Se poate studia modelul MenuDemo de pe github pentru a vedea cum se pot creea meniuri si asocia evenimente acestora.

De exemplu urmatoarea secvență atașează evenimentul de tip acțiune submeniului jMenuItem2:

```
jMenuItem2.setText("First C");
jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
        }
});
```

Funcția care tratează acest eveniment este :

```
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        System.out.println("FirstC");
}
```

Mai multe detalii despre cum lucrează evenimentele vor fi descrise în laboratorul următor.

2.5 Temă

Creeați o aplicație ce comportă fie meniuri fie tab-uri care să lucreze astfel:

- Similar aplicației Notepad. Meniurile vor permite editarea, salvarea și încarcarea unui text pe o componenta textarea.
- Dacă utilizatorul dorește sa lucreze la un al doilea document, el va putea face acest lucru și oricând va putea interschimba între cele două documente deschise.

3 Laboratorul 3. Evenimente

3.1 Evenimente. Aspecte generale

Modelul pentru tratarea evenimentelor în Java este bazat pe folosirea unui frameowrk specific și abstract ce poate fi extins pentru a fi folosit în situații specifice individuale.

Acesta este foarte simplu, fiind format din trei clase (din care una abstractă) și o interfață. În mare, modelul conține convenții de numire pe care programatorul trebuie să le respecte. Urmatoarea imagine descrie modelul Java pentru programarea bazata pe evenimente:

Figura 4. Modelul de evenimente in Java.

Cele mai importante parti din cadrul modelului sunt clasa abstractă java.util.EventObject si interfața java.util.EventListener. Acestea două conțin regulile si convențiile de bază:

- O clasă ce trebuie să fie notificată când un eveniment are loc, se numește ascultător de evenimente (event listener). Un ascultător de evenimente are câte o metodă distinctă pentru fiecare tip de notificare a evenimentelor de care este interesat. Aceste metode au prin conventie urmatoarea semnatura: public void add¡Numele categoriei evenimentului¿Listener(¡Numele categoriei evenimentului¿Listener listener)
- Declarațiile metodelor pentru notificarea evenimentelor sunt grupate pe categorii. Fiecare categorie este reprezentată de o interfață ascultător de evenimente (event listener), ce trebuie sa extindă java.util.EventListener. Prin convenție, un ascultător de evenimente este numit: ¡Numele categoriei evenimentului¿Listener. Orice clasă ce dorește să fie notificată de producerea evenimentelor trebuie să implementeze cel puțin o interfață ascultător.
- Toate aparițiile unui de un eveniment vor fi prinse de un obiect de stare. Clasa din care face parte acest obiect trebuie să fie o subclasă a java.util.EventObject și trebuie să înregistreze cel puțin tipul de obiect care reprezinta sursa evenimentului. O asemenea clasă se numeste clasa eveniment și prin convenție se notează ¡Numele categoriei evenimentului¿Event.
- De obicei (dar nu obligatoriu), o interfață ce ascultă evenimente va face referire la o singură clasa eveniment. Un ascultător de evenimente poate conține mai multe metode de notificare ce primesc un argument din aceeași clasă eveniment.
- De obicei (dar nu obligatoriu), o metodă de notificare a evenimentelor are urmatoarea conventie pentru semnatură: public void ¡evenimentul specific¿(¡Numele categoriei evenimentului¿Event evt).

3.2 Exemplu de caz

Pentru a exemplifica folosirea modelului de tratare a evenimentelor, să presupunem că vrem să scriem un program ce citește un șir de numere introduse de utilizator de la linia de comands, și aplică o procesare asupra lui. Un mod simplu de procesare ar fi însumarea numerelor pe parcursul citirii lor și afișarea sumei finale când citirea a luat sfârșit.

Vom dori ca implementarea solutiei sa fie orientata obiect, adica impartita in clase ce au fiecare responsabilitatile lor specifice. Astfel, vom putea oricand adauga noi operatii asupra sirului de numere, fara a fi nevoie sa aducem modificari la codul existent. Astfel, avem următoarele secvente in cadrul implementarii:

- Citirea șirului de numere de la linia de comandă
- Procesarea sirului de numere
- Pornirea întregului program

Pentru a menține aplicatia simplă, vom fi interesați doar de două evenimente: citirea unui număr și ajungerea la sfârșitul șirului citit.

In primul rând, conform specificațiilor modelului de tratare a evenimentelor, trebuie să declarăm o clasă ce să încapsuleze evenimentele de care suntem interesați. Vom numi aceasta clasa NumberReadEvent, de vreme ce aceasta este acțiunea ce ne interesează. Deasemenea, clasa eveniment trebuie sa moștenească java.util.EventObject. Astfel, clasa de care avem nevoie arata ca mai jos:

```
import java.util.EventObject;

public class NumberReadEvent extends EventObject {

        private Double number;

        public NumberReadEvent(Object source, Double number) {
                super(source);
                this.number = number;
        }

        public Double getNumber() {
                return number;
        }
}
```

Următorul pas este definirea unei interfețe ascultător de evenimente. Această interfață trebuie să definească metode pentru evenimentele ce ne interesează și trebuie să extindă java.util.EventListener. Dupa cum am spus mai sus, evenimentele ce ne interesează sunt citirea unui număr și ajungerea la sfârșitul șirului citit. Astfel, avem urmatoarea implementare pentru interfața ascultător:

```
import java.util.EventListener;

public interface NumberReadListener extends EventListener {
        public void numberRead(NumberReadEvent numberReadEvent);

        public void numberStreamTerminated(NumberReadEvent numberReadEvent);
}
```

Având interfața ascultător definită, putem trece la implementarea uneia sau a mai multor clase ascultător de evenimente. Deocamdată vom implementa o clasă pentru calcularea sumei numerelor din șir. Evident, aceasta clasă trebuie să implementeze interfața NumberReadListener. Pe masură ce evenimentele vor fi prinse de clasa ascultător, un câmp din aceasta clasă va ține suma valorilor primite. În final, vom afișa valoarea campului sumă, când evenimentul de final de șir (numberStreamTerminated) va fi apelat:

```
public class NumberReadListenerImpl implements NumberReadListener {

        Double totalSoFar = 0D;

        @Override
        public void numberRead(NumberReadEvent numberReadEvent) {
                totalSoFar += numberReadEvent.getNumber();
        }

        @Override
        public void numberStreamTerminated(NumberReadEvent numberReadEvent)
           {
                System.out.println("Sum of the number stream: " +
                   totalSoFar);
        }
}
```

Clasa sursă a evenimentului va conține codul pentru citirea șirului de numere, pentru trimiterea evenimentelor către asculăatori și pentru a folosi ascultătorii (adăugarea, eliminarea și urmărirea lor). Pentru simplitarea exemplului, nu vom lua in considerare aspectele legate de multithreading.

```
private Set<NumberReadListener> listeners;

public NumberReader() {
        listeners = new HashSet<NumberReadListener>();
```

```
}
```

În continuare, adăugarea și eliminarea ascultătorilor de evenimente este foarte simplă:

```
public void addNumberReadListener(NumberReadListener listener) {
        this.listeners.add(listener);
}

public void removeNumberReadListener(NumberReadListener listener) {
        this.listeners.remove(listener);
}
```

Chiar dacă nu vom folosi explicit metoda de eliminare a ascultătorilor în acest exemplu, trebuie reținut că modelul de tratare e evenimentelor solicită existența acesteia.

Tipul de colecție folosit, ne permite deasemenea o iterare ușoară printre ascultători, iar apelul metodelor de notificare se va face direct, de vreme ce modelul de tratare a evenimentelor folosește metode de notificare sincron:

```
private void notifyListenersOfEndOfStream() {
        for (NumberReadListener numberReadListener : listeners) {
                numberReadListener.numberStreamTerminated(new
                   NumberReadEvent(this, 0D));
        }
}

private void notifyListeners(Double d) {
        for (NumberReadListener numberReadListener: listeners) {
                numberReadListener.numberRead(new NumberReadEvent(this, d));
        }
}
```

Metoda *notifyListeners* presupune că va primi o valoare Double la apel, iar in cazul metodei *notifyListenersOfEndOfS* valoarea introdusă pentru a semnala finalul șirului de numere va fi ignorată.

În final, avem implementarea codului pentru citirea șirului de numere de la linia de comandă:

```
public void start() {
        Console console = System.console();
        if (console != null) {
                Double d = null;
                do {
                        String readLine =
                        console.readLine ("Enter a number: ",
                           (Object[])null);
                        d = getDoubleValue(readLine);
                        if (d != null) {
                                notifyListeners(d);
                        }
                } while (d != null);
                notifyListenersOfEndOfStream();
        }
}
```

3.3 Temă

Rulați exemplul de mai sus.

Extindeti exemplul pentru a trata alte evenimente ale aceluiasi control tip tree.

Implementati un control de editare fisiere text ce suporta drag-and-drop. Se va reutiliza codul de la laboratorul anterior.

4 Laboratorul 4. Concurenţa in Swing

Un program corect scris in Swing foloseste concurenta pentru a crea interfețe utilizator ce nu ”ingheață” niciodată, adică programul va raspunde mereu interacțiunii cu utilizatorul, indiferent de procesele pe care le execută. Pentru aceasta trebui știut cum tratează framework-ul Swing, firele de executie.

În Swing apar urmatoarele tipuri de fire de executie:

- Fire de executie inițiale, ce rulează codul inițial al aplicației.
- Firul de execuție rezervat expedierii evenimentelor. Majoritatea codului de interacțiune cu framework-ul Swing trebuie executat pe acest fir.
- Fire de execuție ”worker” (sau ”background”), sunt acele fire ce execută în spatele aplicației operațiile ce necesita mult timp de procesare.

Toate aceste fire de execuție sunt puse la dispoziție de runtime sau de framework-ul Swing. Ele trebuiesc folosite pentru a crea aplicații Swing corecte și ușor de intreținut. Pentru aplicații avansate, un program Swing poate crea propriile fire de execuție.

Fire de execuție inițiale Fiecare program are un set de fire de execuție pornesc aplicația. În programele obișnuite, există un singur fir de execuție inițial ce apeleaza metoda principala a clasei de start. Pentru Applet-uri, firele de executie initiale sunt cele ce construiesc obiectul applet si apeleaza metodele sale de initializare si start. Aceste operatii pot avea loc pe un singur fir de executie, pe doua sau chiar pe trei fire diferite, in functie de implementarea platformei Java.

Treaba principala a firelor de execuție inițiale este de a crea un obiect executabil ce inițializează interfața grafică, și programează acel obiect pentru execuția pe firul de expediere a evenimentelor. O dată ce interfața grafică este creată, programul va fi controlat în principal de evenimentele interfeței, fiecare dintre ele cauzând execuția unui task pe firul de execuție rezervat expedierii evenimentelor. Codul executat poate crea noi task-uri pe firul de execuție a evenimentelor (dacă timpul lor de procesare nu afecteaza interfața grafica) sau fire de executie worker (daca timpul de procesare este mai indelungat).

Un fir de execuție inițial programează crearea interfeței grafice prin apelarea javax.swing.SwingUtilities.invokeLater sau javax.swing.SwingUtilities.invokeAndWait. Ambele metode au un singur argument, și anume obiectul executabil ce definește noul task. Singura diferență între ele se observă și din numele lor: invokeLater programează task-ul și se întoarce, pe când invokeAndWait asteapta terminarea task-ului înainte de intoarcere.

Exemplu de apel al unui fir de executie initial:

```
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
                createAndShowGUI();
        }
});
```

Event Dispatch Thread Codul de Swing pentru tratarea evenimentelor rulează pe un fir de execuție special, numit fir de execuție pentru expedierea evenimentelor. De asemenea, majoritatea codului ce apeleaza metode Swing ruleaza tot pe acest fir. Acest lucru este necesar deoarece majoritatea metodelor obiectelor Swing nu sunt ”thread safe”. Aceasta presupune ca la apelarea lor din mai multe fire de execuție apare riscul unor interferențe intre firele de execuție dau a unor erori legate de consistenta memoriei alterând valorile variabilelor într-un mod necontrolat.

Există însă și unele metode sigure la folosirea în cadrul firelor de execuție, acestea fiind marcare ”thread safe” în specificațiile API-ului. Toate celelalte metode ale componentelor Swing trebuie însă apelate în cadrul firului de execuție pentru tratarea/expedierea evenimentelor. Cu toate că programele ce ignoră această regulă pot funcționa corect în marea majoritate a timpului, pot aparea oricând erori neprevăzute ce sunt greu de reprodus.

Este bine să privim codul ce rulează pe firul de execuție pentru tratarea/expedierea evenimentelor ca o serie de mici task-uri. Majoritatea acestora sunt apeluri ale metodelor de tratare a evenimentelor. Alte task-uri pot fi planificate de codul aplicației folosind invokeLater si invokeAndWait. Task-urile ce se executa pe acest fir trebuie sa se incheie repede, in caz contrar, evenimentele netratate rămân în coadă iar interfața grafică nu mai răspunde la acțiunile utilizatorului.

Dacă este nevoie să se determine dacă o secvență de cod rulează pe firul de execuție pentru tratarea/expedierea evenimentelor, se apelează javax.swing.SwingUtilities.isEventDispatchThread.

Fire de executie de tip Worker Cand un program Swing trebuie sa execute un task ce dureaza mult, folosește de obicei un fir de execuție Worker, cunoscut si ca fir de execuție Background. Fiecare task ce ruleaza pe un fir de execuție worker este reprezentat de o instanta a clasei javax.swing.SwingWorker. SwingWorker este o clasa abstracta, de aceea trebuie sa se defineasca o sub-clasa pentru a se putea crea un obiect SwingWorker. Clasele anonime interne sunt de obicei utile pentru crearea de obiecte SwingWorker simplu.

SwingWorker pune la dispoziție o serie de utilitați pentru comunicare și control:

- Subclasa SwingWorker poate defini o metoda ”done” ce este automat apelata pe firul de expediere a evenimentelor atunci cand task-ul ce ruleaza in spate (background task) a fost terminat.
- SwingWorker implementeaza java.util.concurrent.Future. Aceasta interfața permite task-ului ce ruleaza in spate sa returneze o valoare catre celalalt fir de execuție. Alte metode aparținand acestei interfețe permit anularea execuției task-ului, și determinarea faptului daca task-ul s-a terminat de executat sau a fost anulat.
- Task-ul ce ruleaza in spate poate pune la dispoziție rezultate intermediare prin apelarea metodei SwingWorker.publish, ceea ce face ca SwingWorker.process sa fie apelata din firul de execuție pentru tratarea/expedierea evenimentelor.
- Task-ul ce ruleaza in spate poate defini proprietați legate la controale. Modificarile acestor proprietati declanșeaza evenimente, facand ca metodele de tratare a acestora sa fie apelate in cadrul firului de execuție pentru evenimente.

Clasa javax.swing.SwingWorker a fost adaugata platformei Java in versiunea SE 6. Pentru a anula un task ce ruleaza in spate se apeleaza metoda SwingWorker.cancel. Task-ul trebuie sa colaboreze cu propriul proces de anulare.

Sunt doua posibilitati prin care poate face acest lucru:

- Terminandu-se atunci cand primeste o intrerupere.
- Apeland SwingWorker.isCancelled la scurte intervale de timp. Aceasta metoda returneaza true daca acțiunea de anulare a fost apelata pentru acest SwingWorker.

Proprietati de legatura si metode de status SwingWorker suporta proprietați de legatura ce sunt utile pentru comunicarea cu alte fire de executie. Exista doua proprietați predefinite: ”progress” și ”state”. Ca la toate proprietatile de legatura, acestea pot fi folosite pentru a declanșa task-uri pentru tratarea evenimentelor pe firul de execuție rezervat tratarii/expedierii evenimentelor.

Prin implementarea unui ascultator pentru modificarea unei proprietati, un program poate urmari modificari pe orice proprietate legata.

Variabila ”progress” este o valoare int ce poate avea valori intre 0 si 100. Are o metoda setter predefinita (protected SwingWorker.setProgress) și o metoda getter predefinita (public SwingWorker.getProgress).

Variabila legata ”state” indica unde se afla obiectul SwingWorker pe parcursul ”vieții” sale. Aceasta conține o valoare de tip enumerare de tipul SwingWorker.StateValue. Valorile pe care aceasta poate sa le ia sunt:

- PENDING - Starea pe care o are task-ul pe perioada dintre crearea sa pana exact inainte de apelul metodei doInBackground.
- STARTED - Starea task-ului exact dinainte de apelul doInBackground pana exact inainte de apelul metodei done.
- DONE - Starea task-ului in restul perioadei ramase de la apelarea metodei done.

4.1 Temă

Analizați codul din directorul Lab4.

Extindeti exemplul oferit pentru a incarca imagini din alte directoare. Adaugarea imaginilor pe controlul ”toolbar” sa se faca pe n-1 fire, unde n este numarul de core-uri ale mașinii pe care ruleaza aplicația.

Extindeti exemplul notepad anterior pentru a suporta incarcarea unui document folosind SwingWorker. De asemenea cautarea unui text se va face pe n-1 fire.

5 Laboratorul 5. Binding

Designer-ul de forme Java Swing menține Binding de Java Beans conform JSR.

O legatură (Binding) sincronizează două tipuri de proprietăți: sursă și obiectiv. Sursa este de obicei un tip de dată de tip Java Bean așa cum este specificat descrisă aici.

În principiu o clasa Java de tip ”bean” este o clasă ce respectă următoarele:

- este publică nonabstractă
- are un constructor fără parametrii
- are proprietăți publice definite prin getteri și setteri

Exemple de astfel de clase sunt: Beans, Customizer, BeanDescriptor, EventSetDescriptor, Visibility, etc.

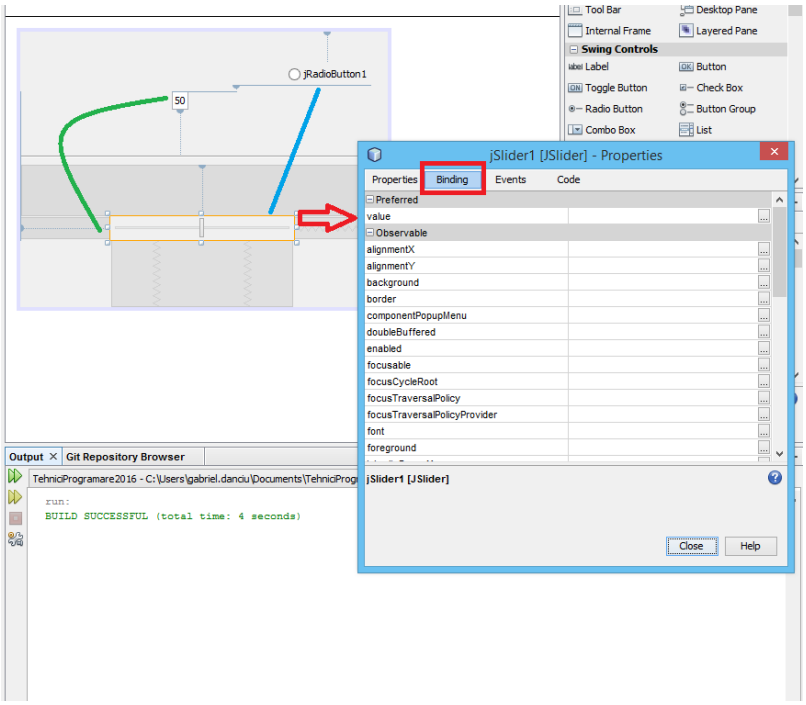

Figura 5. Exemplu de biding

Orice componentă a unui JFrame respectă aceste cerințe iar aceasta implică faptul că se pot creea legături între oricare două asemenea controale. Exemplul BindingHelloWorld prezinta un mod de a creea aceasta legatura intre un slider si un textarea. Totodata aici se prezintă un mod de a realiza conexiunea dintre două proiecte diferite.

În figura 5 se poate vedea cum se creează o legatură ce are ca sursă un slider și destinație un textarea.

Se poate observa în figura 6 cum valoarea slider-ului devine sursă pentru proprietatea text a controlului textfield. Astfel ori de câte ori se modifică valoarea slider-ului, se va modifica și textul din controlul textfield. Reciproca este valabila doar dacă în tabul Advanced se modifică proprietatea ”Update Mode” la ”Always sync (read/write)” .

Legătura s-a făcut între un tip String și un tip Int. Conversia intre tipurile de dată legate se va face automat dacă datele sunt de tipul:

- BigDecimal la String, String la BigDecimal
- BigInteger la String, String la BigInteger
- Boolean la String, String la Boolean
- Byte la String, String la Byte
- Char la String, String la Char
- Double la String, String la Double
- Float la String, String la Float
- Int la String, String la Int
- Long la String, String la BigDecimal
- Short la String, String la Short

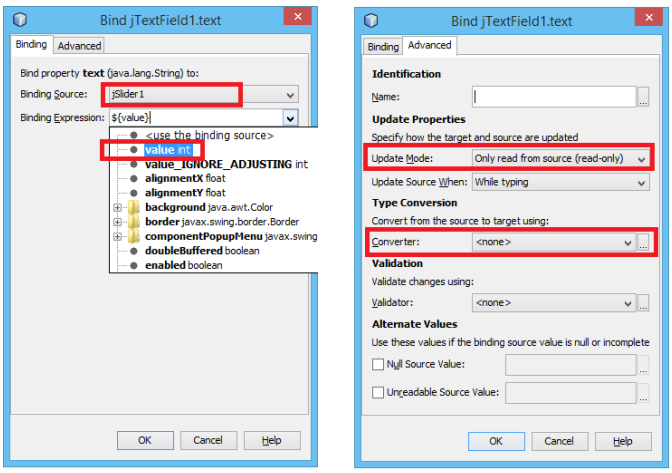

Figura 6. Exemplu de biding

– Int la Boolean, Boolean la Int

Întrebarea firească este dacă nu avem o astfel de conversie automată, sau dorim să adăugăm logică customizabilă atunci când se face transformarea unei informații în alta, ce opțiuni avem? Pentru aceasta există Convertor. Un alt exemplu este legătura dintre jRadiobutton1 si jSlider1, legătură ce suportă o conversie customizată astfel că dacă valoarea slider-ului trece de 50, proprietatea ”selected” a butonului radio devine true, în caz contrar trecând la false.

În figura 6 se poate observa modul în care se setează acest Convertor.

Dacă informația transmisă de la sursă la destinație trebuie validată se poate folosi o clasă Validator.

Legăturile se pot face și între date și componente vizuale. Exemplul BindingList prezită modul în care se poate crea o legătură între o listă de elemente și un jList sau jComboBox.

5.1 Temă

Rulati exemplul din laboratorul 5. Extindeți aplicația din laboratorul 4 astfel încât să creeați un toolbar pentru a controla fontul textului selectat. De asemenea creeați in partea stângă a ecranului principal un alt control care sa indice numărul liniei textului introdus.

6 Laboratorul 6. Comunicare cu baza de date

6.1 Introducere în JDBC

Java DB reprezintă distribuția open source a Apache Derby. Aceasta constă în o serie de standarde, a unui set de caracteristici ce permit dezvoltatorilor Java să lucreze cu o baza de date.

Java Data Objects este un API ce reprezintă un model de abstracție a nivelului de persistență.

Java Database Connectivity (JDBC)

JDBC este un standard pentru crearea conexiunilor la baze de date ce acoperă un larg spectru de tipuri de baze de date. Această tehnologie permite exploatarea conceptului "Write Once Run Anywhere".

Biblioteca JDBC conține clase ce permit lucrul cu baze de date:

- Crearea unei conexiuni cu o bază de date
- Crearea și rularea comenzilor SQL
- Vizualizarea și modificarea rezultatelor

JDBC poate fi utilizat în diferite tipuri de soluții precum aplicații Java, applet-uri, pagini server, servlet-uri s.a.m.d.

6.2 Componente JDBC

Componenta JDBC poate fi implementată în cel puțin patru moduri. JDBC oferă următoarele clase/interfețe care permit lucrul cu bazele de date:

- DriverManager este clasa ce tratează lista de driver-e de baze de date disponibilă. Totodată tratează conectarea unei aplicații java la driver-ul de bază de date corespunzător folosind un subprotocol de comunicare.
- Driver este interfața ce permite comunicarea cu serverul care găzduiește baza de date.
- Connection este interfața ce permite inițierea comunicării cu baza de date. Obiectul ce reprezintă comunicarea vine sub forma unu context de comunicare.
- Statement reprezintă obiectul ce desemnează ce fel de query(CRUD) va fi lansat către baza de date.
- ResultSet conține data provenită din DB ca urmare a query-ului făcut anterior. Ca implementare seamănă cu un iterator ce permite deplasarea între datele obținute ca urmare a rulării query-ului din Statement.
- SQLException permite tratarea erorilor ce pot aparea în comunicarea cu baza de date.

JDBC vine sub forma unui pachet care poate fi downloadat și integrat în proiectul propriu. În funcție de tipul de server aceste pachete pot fi pentru Oracle ori Microsoft SQL Server sau MySQL, etc.

SQL este un limbaj standard ce permite diferite operații dintr-o bază de date cum ar fi operatiile CRUD.

Primul lucru care trebuie făcut este de instala driver-ul JDBC. De exemplu daca lucram cu Microsoft SQL trebuie downloadat pachetul JDBC de aici. Apoi acest pachet poate fi instalat in proiectul dvs. prin adăugarea lui în proprietățile acelui proiect.

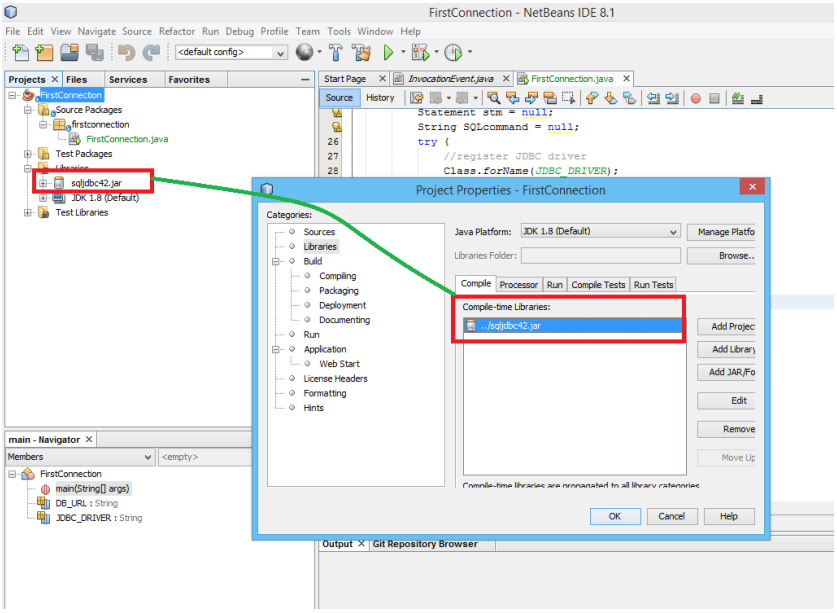

Figura 7. Adăugarea unei biblioteci sqljdbc în proiect

Pentru a folosi corect o conexiune la o bază de date orice aplicație trebuie să respecte următorii pași:

1. se importă pachetele sql
2. se înregistrează driver-ul JDBC
3. se deschide conexiunea
4. se execută un query - comandă SQL
5. se extrage data din result
6. se închide conexiunea

Un exemplu de cod ce realizează acestea este următorul:

```
static final String JDBC_DRIVER =
   "com.microsoft.sqlserver.jdbc.SQLServerDriver";
static final String DB_URL =
   "jdbc:sqlserver://danciugabyserver.database.windows.net:1433;
database=DBFiles;user=gaby;password=Tehnicideprogramare2016;
encrypt=true;trustServerCertificate=false;
hostNameInCertificate=*.database.windows.net;loginTimeout=30;";

...
        try {
                //register JDBC driver
                Class.forName(JDBC_DRIVER);
                //open connection
                conn = DriverManager.getConnection(DB_URL);
                //create SQL statement
                stm = conn.createStatement();
                //create SQL command
                SQLcommand = "Select * from Persons";
                //Execute the command, collect the result
                ResultSet rs = stm.executeQuery(SQLcommand);
                while (rs.next()) {
                        String first = rs.getString("FirstName");
                        String last = rs.getString("LastName");
                        String adress = rs.getString("Address");
                        System.out.println(first);
                        System.out.println(last);
                        System.out.println(adress);
```

```
                }
                //close connections
                rs.close();
                stm.close();
                conn.close();
        } catch (SQLException ex) {
        System.out.println(ex);
        } catch (ClassNotFoundException ex) {
        System.out.println(ex);
        } catch (Exception ex) {
        System.out.println(ex);
        }
```

6.3 Declarații JDBC

Din punct de vedere al implementarii mobul de interacțiune cu o baza de date se face prin declarații (statement). Un exemplu de astfel de declarație este:

```
Statement statemtent = null;
try {
        statemtent = conn.createStatement( "Select * from Students" );
        . . .
}
catch (SQLException e) {
        . . .
}
finally {
        statemtent.close();
}
```

Daca se doreste crearea unor declarații dinamice se poate folosi PreparedStatement. Exemplu:

```
PreparedStatement prepstatement = null;
try {
        String SQL = "Update Students SET age = ? WHERE id = ?";
        prepstatement = conn.prepareStatement(SQL);
        . . .
}
catch (SQLException e) {
        . . .
}
finally {
        . . .
}
```

In cazul in care se doreste apelul unei proceduri stocate se poate folosi ceea ce se numeste un CallableStatement.

```
PreparedStatement pstmt = null;
try {
        String SQL = "Update Students SET age = ? WHERE id = ?";
        pstmt = conn.prepareStatement(SQL);
        . . .
}
catch (SQLException e) {
        . . .
}
finally {
        pstmt.close();
}
```

Un obiect de tip Statement poate fi de trei tipuri:

– execute - Returneaza o valoare booleana cu valoarea true daca se poate obține un obiect de tip ResultSet, altfel returneaza false.

– executeUpdate - Returneaza numarul de randuri afectate de query-ul SQL. De obicei dupa un UPDATE, INSERT sau DELETE se poate afla cate randuri din tabela vizata au fost afectate.
– executeQuery - Returneaza un obiect de tip ResultSet. Aceasta comanda este folosita la interogari de tip SELECT pentru a returna date din baza de date.

Un obiect de tip ResultSet pastreaza un cursor ce pointeaza catre randul actual din tabelul de date returnat. Trebuie menționat ca tabelul poate fi compus din date provenite din mai multe tabele prin intermediul unor interogari de tipul JOIN.

Metodele prezente in interfața ResultSet sunt de trei tipuri:

– Metode pentru a parsa setul rezultat. Sunt metode ce permit cursorului sa navigheze in cadrul setului. Ca si exemple amintim boolean next(), boolean previous(), first(), boolean absolute(int row).
– Metode de prelucrare a datelor. Acestea permit preluarea datelor de la cursorul actual. Exemple: int getInt(String columnName), String getString(String columnName).
– Metode de alterare a datelor. Acestea permit update-uri in baza de date. Exemple: void updateString(int columnIndex, String s) sau void updateRow().

6.4 Tranzactii JDBC

Tranzactiile sunt operațiuni ce permit controlarea modului in care se aplica query-urile ce altereaza baza de date. De exemplu daca intr-un statement avem multiple operatiuni, acestea se vor executa in mod succesiv. Daca insa una din aceste operatiuni esueaza, atunci cele care urmeaza nu se vor mai executa, insa starea bazei este deja alterata de operațiunile anterioare.

Pentru aceasta exista conceptul de tranzactie prezentata in figura 8.

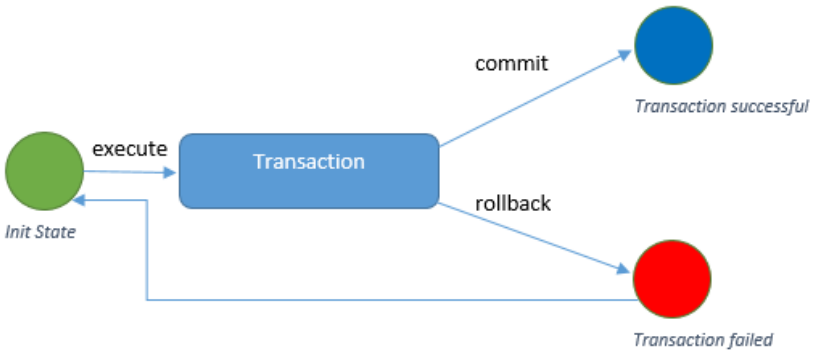

Figura 8. Tranzactii JDBC

Pricipalelec omenzi din cadrul unei tranzatii sunt:

– commit - ceea ce inseamna salvarea starii tranzactiei si executarea query-urilor din cadrul acesteia
– rollback - ceea ce inseamna anularea tuturor instructiunilor din cadrul tranzactiei

Un exemplu de utilizare a unei tranzactii este prezentat mai jos:

```
try{
        ...
        conn = DriverManager.getConnection(DB_URL);
        conn.setAutoCommit(false);
        Statement stmt = conn.createStatement();
        //begin transaction
        String SQL = "INSERT INTO Tabel1  " +
        "VALUES (43, 51, 'ABC')";
        stmt.executeUpdate(SQL);
        //Submit a erronated SQL statement that raises an exception
        String SQL = "INSERT IN Tabel1  " +
```

```
            "VALUES (78, 34, 'DFG', 'This shouldn't be')";
            stmt.executeUpdate(SQL);
            // If there is no error all queries will be performed ok.
            conn.commit();
        }catch(SQLException se){
            // If there is any error we come back to the state before
               transaction.
            conn.rollback();
        }
```

6.5 Tema

Extindeti aplicatia din laboratoarele anterioare pentru a salva si citi documentele dintr-o baza de date. Aplicatia va putea lucra si offline, adica daca se doreste documentele pot fi salvate si citite local. Conexiunea la baza de date presupune si un modul de login, user si password.

7 ORM-Hibernate

Hibernate este un framework open source ce ușurează persistența obiectelor în/din bazele de date. Hibernate realizează atât corelarea claselor din Java cu tabelele din baza de date cât și managementul interogărilor sau tranzacțiilor SQL către bazele de date.

Iată câteva avantaje ale folosirii acestui framework ORM în pofida altora:

1. Hibernate realizează maparea claselor Java folosind fișiere XML adică fără a fi nevoie să folosim alt cod.
2. Dacă se fac schimbări în baza de date, este suficient sa modificăm fișierele XML pentru a păstra maparea inițială.
3. Hibernate nu are nevoie de o altă aplicație server.
4. Minimizează accesul la baza de date prin diverse strategii de preluarea a datelor.
5. Poate trata asocieri complexe între obiectele din baza de date.

În plus Hibernate lucrează cu cele mai populare sisteme de gestiune a bazelor de date precum Microsoft SQL, Oracle, PostgreSQL, DB2, MySQL etc.

7.1 Arhitectura Hibernate

Vom începe prin a prezenta o vedere de ansamblu a acestui framework și a contextului în care acesta este integrat.

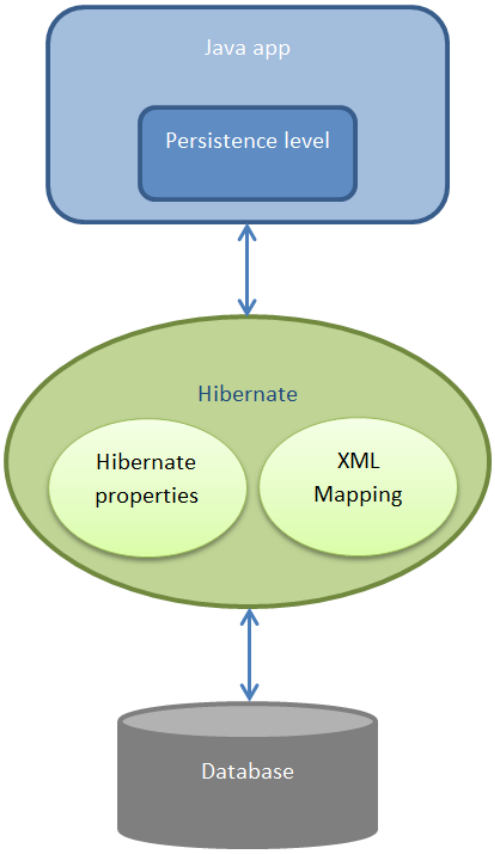

Figura 9. Hibernate: vedere de ansamblu

Figura 9 indică modul în care Hibernate se integrează într-un produs: și anume ca legătură între obiectele Java (substituind nivelul de persistență) și baza de date.

Figura 10 prezintă componentele unui framework Hibernate.

Configurarea este un obiect creat de obicei la inițializarea aplicației.

Acesta conține două tipuri de fișier: de configurare și de mapare. Fișierul de configurare, hibernate.properties sau hibernate.cfg.xml conține datele pentru conexiunea la baza de date. Exemplu:

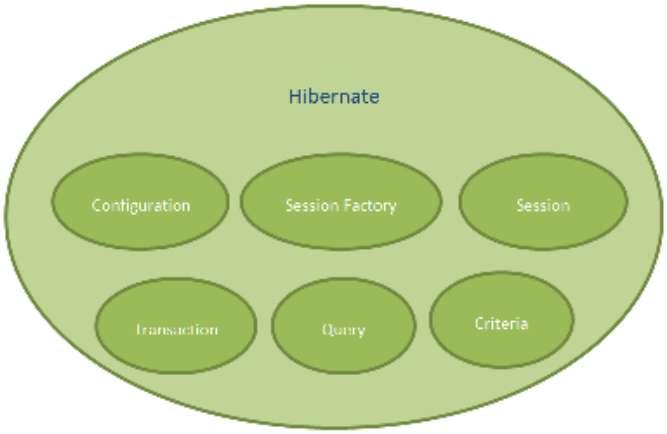

Figura 10. Hibernate: componente principale

```
<?xml version="1.0" encoding="UTF-8"?>
<hibernate-configuration>
<session-factory>
<property
   name="hibernate.connection.driver_class">com.sun.sql.jdbc.sqlserver.SQLServerDriver
<property
   name="hibernate.connection.url">danciugabyserver.database.windows.net:1433</propert
<property name="hibernate.connection.username">gaby</property>
<property
   name="hibernate.connection.password">Tehnicideprogramare2016</property>
<property
   name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
<mapping class="" file="" jar="" package=""
   resource="hibernatesample/hibernate/hbm/hibernate.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

După cum se poate observa, fișierul XML conține noduri care permit setarea tipului de server care găzduiește baza de date, numele acestuia, credențialele de conectare. Pe lângă acestea se poate specifica fișierul XML care va conține detaliile de mapare clase/baza de date. De asemenea tot la partea de configurare trebuie să existe și fișierul de mapare hibernate.hbm care tratează ce obiecte/clase Java sunt corelate cu ce tabele/coloane din baza de date. Exemplu:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
   3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>

<class name="hibernatesample.Employee" table="Employee">
<id name="id" column="id"/>
<property name="first_name" column="first_name"/>
<property name="last_name" column="last_name"/>
<property name="salary" column="salary"/>
</class>
</hibernate-mapping>
```

În acest exemplu se poate observa maparea dintre clasa Employee și tabela Employee din baza de date.

```
class Employee implements Serializable {

        private static final long serialVersionUID = 1L;
        @Id
```

```
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Integer id;
        private String first_name;
        private String last_name;
        private int salary;
        ...
```

Obiectul Configuration este utilizat pentru a crea un obiect SessionFactory care permite instanțiere unui obiect Session. Acest Factory este un obiect ce rulează în propriul fir de execuție în acest fel eliminând problemele de sincronizare a datelor.

Exemplu:

```
private static SessionFactory factory;
...
Configuration configuration = new Configuration();
configuration.configure("hibernate.cfg.xml");
factory = configuration.buildSessionFactory();
```

Un obiect Session este folosit pentru a realiza conexiunea cu baza de date ori de câte ori este nevoie de o interacțiune cu aceasta. Obiectele persistente în memorie sunt salvate sau populate în și din baza de date prin intermediul Session.

```
Session session = factory.openSession();
```

Un obiect Transaction este unitatea de lucru pentru a crea tranzacții cu baza de date. Aceste tranzacții sunt tratate de către un manager JDBC sau JTA.

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

```
 Transaction tx = null;
Integer employeeID = null;
try{
        tx = session.beginTransaction();
        Employee employee = new Employee();
        employee.setId(4);
        employee.setFirst_name(fname);
        employee.setLast_name(lname);
        employee.setSalary(salary);
        employeeID = (Integer) session.save(employee);
        tx.commit();
```

Update-ul in baza de date se face la apelul funcției commit.

7.2 Tema

Extindeți aplicația pentru ca persistența obiectelor (exemplu documentele tratate de aplicație) în baza de date să se facă printr-un ORM.

8 Principiile Solid

9 Desing Patterns