Aplicații Integrate pentru Întreprinderi

Laborator 4

05.11.2012 / 08.11.2012

Dezvoltarea unei interfețe grafice cu utilizatorul folosind Java FX

Scopul laboratorului îl reprezintă familiarizarea cu unele facilități oferite de platforma JavaFX pentru dezvoltarea de interfețe grafice precum şi înțelegerea arhitecturii pe baza căreia a fost construită aceasta.

1. Ce este JavaFX ?

JavaFX reprezintă o tehnologie pusă la dispoziție programatorilor Java spre a implementa aplicații Internet (*eng.* RIA – rich Internet applications) având un comportament consistent pe mai multe platforme diferite. În cadrul JavaFX a fost dezvoltat un API extins pentru aplicații multimedia avansate şi motoare grafice care facilitează crearea unor aplicații pentru întreprinderi bazate pe date. Fiind bazat pe limbajul de programare Java, va fi uşor de utilizat, optimizând atât costurile legate de investiții, cât şi complexitatea sistemelor, întrucât atât componenta server (nivelul de logică a aplicației şi de acces la date) cât şi componenta client (nivelul de prezentare) vor fi implementate utilizând acelaşi limbaj de programare.

Câteva dintre caracteristicile JavaFX sunt următoarele [1]:

- **integrare cu Java 7** – începând cu versiunea Java 7 update 6, SDK-ul JavaFX[1] este integrat în distribuția limbajului de programare, astfel că instalarea lor se face împreună[2]; JavaFX beneficiază de API-uri Java cu toate caracteristicile acestora (parametrizarea claselor cu tipuri de date, adnotări, suport pentru aplicații distribuite), permițând o interfațare cu alte limbaje de programare care folosesc Java Virtual Machine, precum JRuby sau Scala;
- **motor grafic îmbunătățit**, optimizat pentru unități de procesare grafică (*eng.* GPU – Graphic Processing Unit), bazat pe Prism, un accelerator implementat ca o bandă de asamblare şi Glass, utilitar pentru dezvoltarea sistemelor de ferestre; se obțin aplicații complexe care rulează rapid datorită integrării la nivel hardware;
- **limbajul declarativ de adnotare FXML**, bazat pe XML, utilizat pentru definirea interfeței cu utilizatorul, având avantajul că aplicația nu trebuie recompilată atunci când se modifică dispunerea elementelor grafice;

[1] În prezent, SDK-ul Java 7 a ajuns la distribuția update 9, iar cel de JavaFX la versiunea 2.2.3.
[2] O versiune JavaFX va fi disponibilă pentru instalare separată în vederea integrării cu Java 6, fiind disponibilă până în noiembrie 2012.

- **motor multimedia îmbunătățit**, care implementează redarea de conținut multimedia folosind tehnologia GStreamer, caracterizată prin stabilitate şi latență scăzută;
- **o componentă WebKit HTML** ce permite integrarea de pagini Internet în cadrul aplicațiilor JavaFX[3];
- **paletă largă de controale pentru interacțiunea cu utilizatorul**, incluzând grafice, tabele, meniuri şi panouri[4];
- **un utilitar pentru integrarea claselor aplicației cu bibliotecile**, obținându-se distribuții care pot fi rulate pe orice platformă;
- **disponibilitate pe majoritatea platformelor** (Windows, Linux, MacOS) începând cu JavaFX 2.2, astfel încât se oferă un comportament consistent atât pentru programatori, cât şi pentru utilizatori.

De asemenea, sunt puse la dispoziția utilizatorilor şi numeroase exemple de aplicații folosind tehnologia JavaFX, precum şi o documentație extinsă[5].

Aplicații JavaFX pot fi rulate fie direct pe o anumită maşină, unde se află „instalată”[6] (accesând fişierul .jar care conține programul în sine), fie poate fi accesată prin intermediul unui navigator (*eng.* browser), fiind integrată în cadrul unei pagini Internet[7]. Totodată, se poate ca sistemul informatic să fie descărcat de la o anumită locație, rulând apoi pe maşina respectivă (modul WebStart[8]).

2. Arhitectura JavaFX

Platforma JavaFX are de o arhitectură bazată pe o stivă de componente[9], constituind motorul ce rulează codul propriu-zis (Quamtum Toolkit). Acesta cuprinde motorul grafic de performanță înaltă (Prism), sistemul de ferestre (Glass), un motor pentru redarea conținutului multimedia şi un motor pentru integrarea conținutului Internet.

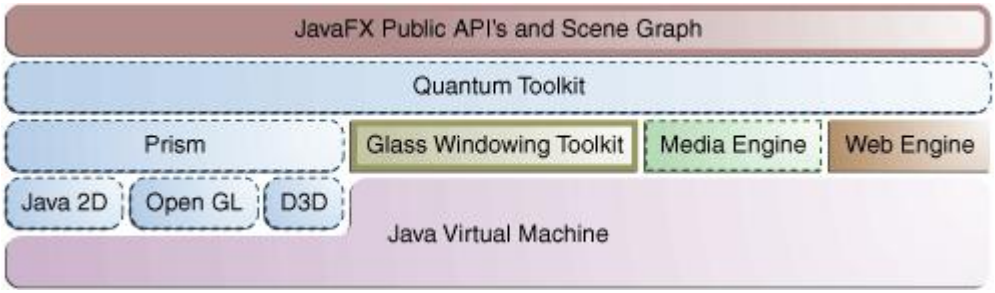


Figura 1 – Arhitectura JavaFX

[3] Prin utilizarea Prism, redarea unui astfel de conținut va beneficia şi de accelerarea hardware.
[4] Este disponibil şi un API pentru dezvoltarea de noi controale pentru interfața cu utilizatorul care pot fi utilizate apoi de întreaga comunitate de programatori JavaFX.
[5] Programatorii pot beneficia de funcționalitățile utilitarului javadoc pentru a dezvolta documentații ale propriilor implementări în format HTML.
[6] O astfel de opțiune este preferată în cazul în care aplicația nu are nevoie de acces Internet sau maşina pe care rulează nu dispune de o astfel de conexiune.
[7] Interacțiunea cu elementele paginii Internet se poate face prin intermediul limbajului JavaScript.
[8] Aplicația poate fi pornită prin accesarea fişierului JNLP.
[9] Componentele JavaFX funcționează în mod transparet, nefiind accesibile propriu-zis programatorilor.

La nivelul cel mai înalt al arhitecturii JavaFX se află **graful de scene** (*eng.* Scene Graph), structură ierarhică de noduri[10] ce conține elementele vizuale ale interfeței grafice cu utilizatorul, care poate trata evenimente de intrare şi care poate fi redată. Un element este identificat în mod unic, fiind caracterizat printr-o clasă de stil şi un volum care îl delimitează. Fiecare nod are un părinte[11], putând avea nici unul sau mai mulți copii. De asemenea, pentru un astfel de element pot fi definite efecte, opacitatea, transformări, mecanisme de tratare pentru diferite evenimente (ce țin de interacțiunea cu utilizatorul) precum şi starea aplicației. Spre diferență de Swing sau AWT (Abstract Window Toolkit), JavaFX conține şi primitive pentru elemente grafice[12], pe lângă mecanisme de dispunere a conținutului, controale, imagini şi obiecte multimedia.

API-ul javafx.scene permite construirea următoarelor tipuri de conținuturi

- noduri: forme 2D şi 3D, imagini, conținut multimedia şi conținut Internet, text, controale pentru interacțiunea cu utilizatorul, grafice, containere;
- stări: transformări (poziționări şi orientări ale nodurilor), efecte vizuale;
- efecte: obiecte care modifică aspectul nodurilor (mecanisme de estompare, umbre, reglarea culorilor).

Platforma JavaFX pune la dispoziție şi **un set de API-uri** care permit implementarea unor aplicații Internet[13] într-un mod flexibil, exploatând capabilitățile limbajului de programare Java pentru a dezvolta intuitiv funcționalități referitoare la conținut multimedia. Se permite de asemenea integrarea cu alte limbaje dinamice (JRuby, Groovy, JavaScript) din care facilitățile JavaFX pot fi accesate. Mecanismul de tipare permite evaluare leneşă, inclusiv pentru expresii şi secvențe de expresii prin intermediul sintaxei. Colecțiile din Java sunt îmbogățite căci interfața cu utilizatorul este conectată cu modele de date, astfel ca diferitele operații să se reflecte în conținutul aplicației.

În **sistemul grafic** din JavaFX pot fi implementate obiecte 2D şi 3D care sunt randate software, atunci când hardware-ul nu dispune de accelerare grafică, prin intermediul unor benzi de asamblare:

- **Prism** este folosit pentru operații de redare / rasterizare a conținutului (scene JavaFX)[14] folosind DirectX 9 (Windows XP / Vista), DirectX 11 (Windows 7), OpenGL (Mac, Linux, sisteme încorporate), Java 2D[15] (dacă hardware-ul nu dispune de accelerare grafică);
- **Quantum Toolkit** reprezintă un element de interfață între Prism şi sistemul de ferestre Glass pe care le face disponibile nivelului JavaFX, ocupându-se şi de prioritatea firelor de execuție ce tratează redarea raportată la tratarea evenimentelor de intrare/ieşire.

[10] Prin termenul de nod se desemnează un element din graful de scene.
[11] Excepție de la această regulă o face nodul rădăcină.
[12] Se pot crea animații folosind aceste elemente grafice, folosind metodele puse la dispoziție de API-urile `javafx.animation`.
[13] Platforma JavaFX se bazează pe numeroase standarde Internet, cum ar fi CSS pentru stiluri sau ARIA pentru specificări referitoare la accesibilitate.
[14] Prism suportă atât redare hardware (când sistemul pe care rulează aplicația dispune de acceleratoare grafice) cât şi redare software (când echiparea maşinii în cauză este insuficientă). Prism implementează şi redarea de obiecte 3D.
[15] Atunci când nu poate fi folosită accelerarea grafică la nivel hardware, se utilizează Java2D datorită răspândirii ei la nivelul JRE, mai ales când scena conține obiecte 3D. Totuşi, în acest caz, performanțele sunt mai reduse decât în cazul în care se foloseşte randarea la nivel hardware.

În arhitectura JavaFX, la nivelul cel mai scăzut se află **Glass – un sistem de ferestre**, în fapt o interfață între API-ul JavaFX şi sistemul de operare care gestionează ferestrele, mecanisme de cronometrare, suprafețe, evenimentele[16] de intrare / ieşire.

Platforma JavaFX rulează pe mai multe fire de execuție (minim 2):

- firul de execuție JavaFX utilizat pentru a gestiona scenele care fac parte din ferestrele care sunt afişate[17];
- firul de execuție Prism folosit pentru redare, separat de gestiunea evenimentelor de intrare / ieşire – permițându-se ca un cadru să fie afişat în timp ce alt cadru este procesat[18]; acesta poate asocia alte fire de execuție folosite pentru rasterizare, optimizând procesul de randare propriu-zis;
- un fir de execuție pentru conținut multimedia, ce rulează în fundal, utilizat pentru sincronizarea cadrelor prin colaborare cu firul de execuție JavaFX.

Sincronizarea se face prin intermediul mecanismului **Pulse**. Un puls este un eveniment care indică grafului de scene JavaFX faptul că trebuie sincronizată starea elementelor pe care le conține prin intermediul Prism. El se declanşează la maxim 60 de cadre pe minut sau oricând sunt redate diferite animații[19], actualizând starea elementelor din graful de scene, evenimentele fiind tratate asincron. Se permite astfel ca evenimentele să fie tratate în funcție de puls[20]. Evenimentele din cadrul unui puls sunt executate de sistemul de ferestre Glass prin intermediul unui sistem de cronometrare (de rezoluție înaltă) nativ.

JavaFX oferă suport pentru multimedia prin API-urile `javafx.scene.media` care implementează atât conținut auditiv[21] cât şi vizual[22]. Sunt implementate trei componente: obiectul Media care reprezintă fişierul, obiectul MediaPlayer care redă conținutul acestuia şi obiectul MediaView care este nodul ce afişează utilizatorilor obiectul multimedia propriu-zis.

JavaFX include şi un navigator care permite redarea conținutului Internet prin intermediul API-ului său. Componenta WebEngine se bazează pe WebKit, navigator open-source care suportă HTML5, CSS, JavaScript, DOM şi SVG, oferind o funcționalitate asemănătoare cu cele mai multe produse de acelaşi tip

[16] Spre diferență de AWT care foloseşte o coadă de evenimente de intrare/ieşire proprie, JavaFX utilizează coada de evenimente a sistemului de operare gazdă pentru a gestiona modul de alocare al acestora la firele de execuție. O altă caracteristică proprie este rularea sistemului de ferestre Glass pe acelaşi fir de execuție ca şi aplicația JavaFX (în AWT exista un fir de execuție specific bibliotecii şi un fir de execuție specific Java, ceea ce crea o serie de probleme).

[17] Un graf de scene poate fi creat într-un fir de execuție separat, însă în momentul în care elementul său rădăcină este legat de un obiect curent, acesta va fi tratat de firul de execuție JavaFX. Astfel, se permite ca utilizatorii să creeze în fundal scene complexe, menținându-se constantă viteza de redare a scenei curente.

[18] Procesarea concurentă reprezintă un avantaj în special în contextul sistemelor multiprocesor.

[19] Un puls poate fi generat chiar şi atunci când nu sunt redate animații, însă există modificări în graful de scene.

[20] Actualizarea dispunerii conținutului sau a stilului sunt legate de asemenea de puls. Dacă aceste modificări sunt numeroase (în cazul modificărilor din graful de scene), există riscul ca performanța să fie redusă semnificativ, motiv pentru care ele sunt parcurse doar o singură dată în cadrul unui puls. Programatorii pot forța parcurgerea cozii de modificări (dispunere conținut / stil) pentru a deveni vizibile înaintea unui puls.

[21] Sunt implementate formatele .mp3, .aiff şi .wav.

[22] Este suportat formatul .flv.

(redare conținut HTML local şi aflat la o anumită locație, implementare istoric cu navigare înainte şi înapoi în cadrul acestuia, reîncărcarea conținutului, aplicarea unor efecte, editarea conținutului HTML, execuția de comenzi JavaScript şi tratarea evenimentelor). La nivelul componentei care integrează navigatorul sunt definite două clase: `WebEngine` care oferă funcționalități de bază caracteristice unui navigator şi `WebView`[23] care încapsulează un obiect `WebEngine`, încorporând conținut HTML în scena aplicației, oferind atribute şi metode pentru aplicarea de efecte şi transformări.

JavaFX permite stilizarea interfeței cu utilizatorul folosind foi de stil (*eng.* CSS – Cascading Style Sheets), deci fără a modifica codul sursă al aplicației. Acestea pot fi aplicate[24] asupra oricărui obiect de tip `Node` din graful de scene. Fişierele conținând foi de stil pot fi parsate de orice utilitar, chiar dacă acesta nu este integrat cu JavaFX. Integrarea JavaFX / CSS poate fi utilizată şi pentru pagini HTML. Toate proprietățile JavaFX sunt prefixate cu şirul `-fx-` chiar şi pentru cele compatibile CSS HTML, datorită faptului că semantica poate fi diferita în acest caz pentru valorile corespunzătoare unor atribute.

Controalele pentru interfața cu utilizatorul disponibile prin API-ul JavaFX (`javafx.scene.control`) sunt implementate folosind noduri din graful de scene. Ele pot fi particularizate folosind foile de stil.

Controalele pot fi grupate în containere sau panouri în mod flexibil folosind mai multe tipuri de dispunere (*eng.* layout). API-ul JavaFX Layout defineşte mai multe clase de tip container predefinite:

- `BorderLayout` dispune nodurile conținute în regiunile de sus, de jos, dreapta, stânga sau centru;
- `HBox` îşi aranjează conținutul orizontal pe un singur rând;
- `VBox` îşi aranjează conținutul vertical pe o singură coloană;
- `StackPane` utilizează o stivă de noduri afişând elementele unele peste altele
- `GridPane` permite utilizatorului să îşi definească un tabel (format din rânduri şi coloane) în care să poată fi vizualizate elementele conținute;

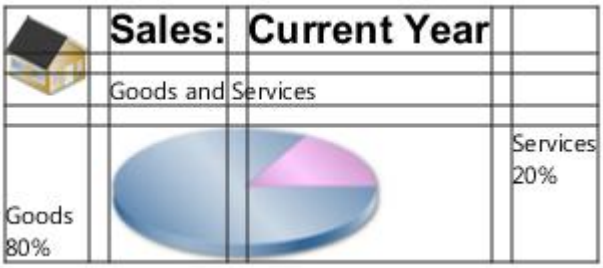


Figura 2 – Dispunerea de tip GridPane

- `FlowPane` dispune elementele fie orizontal, fie vertical, în funcție de limitele specificate de programator (lungime pentru dispunere orizontală, respectiv înălțime pentru dispunere verticală);
- `TilePane` plasează nodurile conținute în celule de dimensiuni uniforme;
- `AnchorPane` oferă programatorilor posibilitatea de a defini noduri ancoră (referință) în funcție de colțurile de jos / sus, din stânga / dreapta sau raportat la centrul containerului sau panoului.

[23] `WebView` reprezintă o extensie a clasei `Node`.

[24] Aplicarea se face în mod asincron şi se poate realiza inclusiv la runtime, permițând modificarea conținutului în mod dinamic. Este implementată specificația W3C CSS versiunea 2.1.

Diferitele moduri de dispunere pot fi imbricate în cadrul unei aplicații JavaFX pentru a se obține funcționalitatea dorită.

Fiecare nod din graful de scene JavaFX poate suferi **transformări** (2D/3D) în sistemul de cordonate Oxyz folosind pachetul javafx.scene.transform:

- `translate` – mută un nod dintr-un loc în altul de-a lungul planurilor x, y, z relativ la poziția sa inițială;
- `scale` – redimensionează un nod pentru a apărea fie mai mare, fie mai mic în planurile x, y, z (în funcție de factorul de scalare);
- `shear` – roteşte o axă astfel încât axele x sau y să nu mai fie perpendiculare coordonatele nodului fiind modificate conform specificațiilor;
- `rotate` – roteşte un nod în jurul unui punct, denumit pivot al scenei;
- `affine`[25] – realizează o mapare (liniară) dintr-un sistem de coordonate 2D/3D în alt sistem de coordonate 2D/3D păstrând caracteristici ale dreptelor precum paralelismul sau ortogonalitatea.

Îmbogățirea aplicațiilor JavaFX (în special a celor de timp real) se face prin efecte vizuale care operează la nivel de pixel al imaginii[26], cum ar fi `DropShadow` (adaugă o umbră în spatele conținutului respectiv), `Reflection` (crează o versiune reflectată a conținutului respectiv, dispunând-o sub acesta) şi `Lighting` (simulează o sursă de lumină care operează asupra unui continut, dând un aspect mai realistic, 3D, asupra unui obiect plat).

3. Structura unei aplicații JavaFX

O aplicație JavaFX trebuie să implementeze o clasă care extinde `javafx.application.Application`, metoda `start()` fiind apelată la execuția acesteia. Containerul interfeței cu utilizatorul se exprimă prin clase de tip `Stage`[27] şi `Scene`[28].

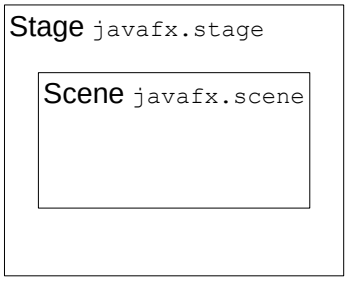


Figura 3 – Containere într-o aplicație JavaFX

În exemplul de mai jos, aplicația JavaFX poartă numele `BookStore` şi extinde clasa `Application`. Metoda `start` primeşte un argument de tip `Stage` care reprezintă container-ul de nivel înalt. În cadrul acesteia se defineşte o scenă având ca element rădăcină un element `VBox` (care identifică un mod de dispunere pe verticală a nodurilor) şi dimensiuni obținute din cele ale dispozitivului pe care va fi vizualizată aplicația. Elementele din scenă vor fi adăugate prin metoda `addAll`, apelată din contextul obiectului care întoarce lista nodurilor copil al elementului radăcină.

[25] Această metodă ar trebui să fie folosită împreună cu clasele de transformare `Translate`, `Scale`, `Rotate`, `Shear` mai degrabă decât să fie apelate direct.
[26] Toate nodurile din graful de scene sunt randate ca imagini, iar efectele sunt aplicate asupra acestor imagini.
[27] Clasa `Stage` reprezintă container-ul de nivel înalt.
[28] Clasa `Scene` reprezintă container-ul pentru toate elementele.

```
public class BookStore extends Application {
    private Stage   applicationStage;
    private Scene   applicationScene;
    private double  sceneWidth, sceneHeight;

    @Override
    public void start(Stage mainStage) {
        applicationStage = mainStage;
        mainStage.setTitle(Constants.APPLICATION_NAME);
        Dimension screenDimension =
                  Toolkit.getDefaultToolkit().getScreenSize();
        sceneWidth  = Constants.SCENE_WIDTH_SCALE*screenDimension.width;
        sceneHeight = Constants.SCENE_HEITH_SCALE*screenDimension.height;
        applicationScene = new Scene(new VBox(), sceneWidth, sceneHeight);
        ((VBox) applicationScene.getRoot()).getChildren().addAll(...);
        mainStage.setScene(applicationScene);
        mainStage.show();
    }
```

Metoda `main()` nu este necesară atunci când fişierul .jar corespunzător aplicației JavaFX este creat folosind utilitarul JavaFX Packager, care include şi un mecanism pentru lansarea aplicației în cadrul acesteia (JavaFX launcher). Totuşi, metoda `main()` va trebui implementată atunci când nu se utilizează acest utilitar (IDE-uri care nu au integrat JavaFX[29]).

4. Construirea de interfețe grafice cu ajutorul controalelor JavaFX

Controalele care gestionează interfața cu utilizatorul în JavaFX sunt noduri în graful de scene, putând fi particularizate fie folosind CSS[30] fie utilizând clasa `Skin`. Totodată, ele pot fi integrate cu uşurință cu alte aplicații Java[31]. Clasele care implementează controale din JavaFX se găsesc în pachetul `javafx.scene.control`. Pe lângă controalele clasice (existente în AWT şi Swing) au mai fost dezvoltate şi alte componente, cum ar fi `Accordion`, `ColorPicker`, `Chart`[32], `Pagination` sau `TitledPane`.

Fiind derivate din clasa `Node`, controalele pentru interfața cu utilizatorul pot fi integrate în diferite animații, efecte[33], transformări sau tranziții în cadrul procesului de randare a scenei.

Clasa `Control` poate fi extinsă definindu-se un obiect care corespunde nevoilor utilizatorilor.

Fiecare control oferă atribute şi metode (suplimentare față de cele din clasa `Control`) spre a oferi suport intuitiv pentu interacțiunea cu utilizatorul.

---

[29] Singurul IDE care este integrat cu JavaFX este Netbeans 7.2.

[30] Utilizarea foilor de stil (CSS) în aplicațiile JavaFX este similară ca şi în HTML întrucât folosesc aceeaşi specificație CSS. Pentru fiecare obiect instanță a unui control se poate apela metoda `setStyle` primind ca parametru un şir de caractere de forma atribut:valoare, unde atribut este denumirea elementului de stil (prefixat cu `-fx-`), iar valoare reprezintă caracteristica asociată, respectând o semantică specifică. Un fişier .css care specifică formatul în care sunt vizualizate diferite elemente este asociat unei scene prin metoda `scene.getStyleSheets().add("...css")`.

[31] Integrarea cu o aplicație Swing se face creând un obiect `Scene` la care se adaugă toate nodurile (având un mod de dispunere asociat), acesta fiind ulterior adăugat la containerul corespunzător din aplicația respectivă. Chiar şi în acest caz, elementele JavaFX sunt randate folosind Prism.

[32] În `javafx.scene.chart` sunt definite mai multe tipuri de grafice (area, bubble, line, pie, scatter) acestea putând conține mai multe seturi de date.

[33] Efectele vizuale disponibile în pachetul `javafx.scene.effect` sunt umbrirea, iluminarea sau estomparea imaginii.

Figura 4 – Tipuri de controale definite în JavaFX

Un obiect de tip `Label` poate primi ca argument textul asociat[34] precum şi imaginea (obiect de tip `ImageView`) care va fi afişată împreună cu acesta. Parametrii pot fi specificați şi prin intermediul unor metode:

- `setText(String text)`
- `setGraphic(Node graphic)`

Şi obiectele de tip `Button` suportă aceleaşi metode, întrucât sunt derivate din clasa `Labeled`. Când se foloseşte atât text cât şi imagine pentru un buton, distanțarea dintre ele se poate face folosind metoda `setGraphicTextGap`.

Aceleaşi metode pot fi folosite pentru obiecte de tip `RadioButton`, însă acestea trebuie grupate în cadrul unui obiect de tip `ToggleGroup`, identificarea obiectului selectat făcându-se prin metoda `getSelectedToggle`.

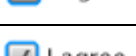
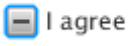
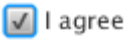
Figura 5 – Modele de obiecte de tip `Label`, `Button` şi `RadioButton`

[34] Culoarea cu care va fi afişat textul respectiv se stabileşte prin metoda `setTextFill()`, iar indicarea unui font se face prin metoda `setFont()`.

Un tip particular de butoane asemănător cu `RadioButton` este `ToggleButton`, fiind de asemenea asociat unui grup[35].

Obiectele de tip `CheckBox` nu pot fi grupate în obiecte `ToggleGroup` în care doar o selecție să fie posibilă. Starea unui obiect `CheckBox` poate fi determinată (caz in care poate fi selectat sau deselectat) sau nedeterminată, proprietatea specificându-se prin metoda `allowIndeterminate`.

Tabelul 1 – Stările unui obiect de tip `CheckBox`

	INDETERMINATE=false[36]	INDETERMINATE=true[37]
SELECTED=false		
SELECTED=true		

Prin intermediul obiectelor de tip `CheckBox` se oferă posibilitatea selecției între un număr relativ mic de opțiuni. Elementele din cadrul unui astfel de obiect (create prin metoda `FXCollections.observableArrayList`) pot fi specificate fie atunci când obiectul este construit, fie prin metoda `setItems`. Delimitarea dintre opțiuni se poate realiza utilizând obiecte de tip `Separator`. De asemenea, nodul poate fi însoțit de o explicație suplimentară, folosind metoda `setTooltip` (care primeşte drept parametru un obiect de tipul `Tooltip`).

Clasele `TextField` şi `PasswordField`, derivate din clasa `TextInput` oferă posibilitatea de a primi un text de la utilizator[38] pe care îl poate afişa. Crearea unui astfel de obiect se face cu sau fără argument. Dimensiunea câmpului text este precizată prin metoda `setPrefColumnCount` în timp ce textul descriptiv indicând semnificația conținutului[39] este indicat prin metoda `setPromptText`. Metodele puse la dispoziție de aceste clase sunt:

- `copy()` – transferă textul selectat într-o zonă de memorie, menținându-l în obiectul curent;
- `cut()` – transferă textul selectat într-o zonă de memorie, nemenținându-l în obiectul curent;
- `paste()` – transferă conținutul zonei de memorie în obiectul curent, înlocuind textul selectat.

Elementele unui obiect de tip `ScrollBar` sunt cursorul, bara culisantă precum şi butoanele stâng şi drept. Câteva dintre metodele caracteristice acestui tip de element sunt `setMin` şi `setMax` pentru a exprima limitele între care cursorul se mişcă, `setValue` pentru a specifica poziția curentă a cursorului, `setOrientation` pentru a distinge între tipurile orizontal şi vertical. Mutarea cursorului (într-o anumită direcție) cu o singură unitate (specificată prin proprietatea `UNIT_INCREMENT`) se face accesând butoanele din stânga sau din dreapta. Valoarea poate fi ajustată prin atributul `BLOCK_INCREMENT`.

[35] Adăugarea în cadrul unui grup se face prin metoda `setToggleGroup`.
[36] În cazul în care proprietatea INDETERMINATE are valoarea false, starea unui obiect `CheckBox` poate fi selectat sau deselectat.
[37] În cazul în care proprietatea INDETERMINATE are valoarea true, starea unui obiect `CheckBox` poate fi selectat, deselectat sau nedefinit.
[38] Diferența dintre clasele `TextField` şi `PasswordField` este că în cazul `TextField` conținutul e afişat în timp ce pentru `PasswordField` conținutul nu e afişat. De asemenea, o practică frecventă pentru obiecte de tip `PasswordField` este ştergerea conținutului său după ce a fost realizată autentificarea
[39] Diferența dintre conținutul introdus de utilizator într-un obiect `TextField` / `PasswordField` şi textul explicativ este că doar valoarea introdusă poate fi obținută prin metoda `getText`.

Obiectele de tip `ScrollPane` oferă o vizualizare a controalelor ce asigură interfața cu utilizatorul folosind elemente derulante (`ScrollBar`). Conținutul (nodul[40]) unui astfel de obiect este specificat prin intermediul metodei `setContent`. Se pot specifica politici de afişare a elementelor derulante, atât pentru verticală şi pentru orizontală prin metodele `setHbarPolicy` / `setVbarPolicy` Parametrii pe care îi pot lua aceste metode sunt definite în clasa `ScrollBarPolicy`: `ALWAYS`, `NEVER`, `AS_NEEDED`. Componentele din cadrul unui obiect `ScrollPane` pot fi redimensionate pentru a corespunde spațiului existent (metodele `setFitToWidth` / `setFitToHeight`)[41].

Elementele `ListView` sunt utilizate pentru vizualizarea unei liste care conține mai multe obiecte, astfel încât pentru inspectarea lor este necesară existența unei bare derulante. Componentele sale pot fi specificate atunci când obiectul este construit sau prin metoda `setItems` (se foloseste constructorul `FXCollections.observableArrayList`). Dimensiunile sunt specificate prin `setPrefHeight`, respectiv `setPrefWidth`, valorile fiind exprimate în pixeli. Clasele `SelectionModel` şi `FocusModel` sunt folosite pentru a identifica selecția şi focusul din cadrul listei[42]:

- `getSelectionModel().getSelectedIndex()` – întoarce indexul elementului selectat în mod curent;
- `getSelectionModel().getSelectedItem()` – întoarce elementul selectat;
- `getFocusModel().getFocusedIndex()` – întoarce indexul elementului care deține focusul în mod curent;
- `getFocusModel().getFocusedItem()` – întoarce elementul focusat.

Pentru a permite selecția multiplă se va folosi metoda `setSelectionMode` (având ca parametru `SelectionMode.MULTIPLE`), elementele selectate obținându-se prin proprietățile `selectedItems` / `selectedIndices`.

JavaFX pune la dispoziție programatorilor şi clase pentru a reprezenta datele într-o formă tabelară: `TableView`, `TableColumn` şi `TableCell`. Popularea implică definirea unui model pentru tabel şi asocierea unei fabrici pentru celule. Modelul de date este o clasă ale cărei atribute au tipul `SimpleStringProperty` şi pentru care se definesc obiecte de tip getter şi setter. Pentru fiecare coloană existentă în cadrul tabelului se va specifica, pe lângă nume şi dimensiune şi fabrica pentru valorile pe care le va conține:

```
for (int currentIndex=0; currentIndex<attributes.size(); currentIndex++) {
    TableColumn column = new TableColumn(attributes.get(currentIndex));
    column.setMinWidth((int)(sceneWidth / attributes.size()));
    column.setCellValueFactory(
    new PropertyValueFactory<Entity,String>(attributes.get(currentIndex)));
    tableContent.getColumns().addAll(column);
}
```

[40] Un obiect de tip `ScrollPane` poate conține un singur nod. Dacă se doreşte adăugarea mai multor noduri, se vor folosi containere cu dispunerea conținutului sau clasa `Group`. De asemenea, metoda `setPannable` poate fi utilizată (cu argumentul `true`) pentru ca obiectul conținut să fie vizualizat odată cu mişcarea mouse-ului.

[41] Implicit, proprietățile `FIT_TO_WIDTH` şi `FIT_TO_HEIGHT` au valoarea false. De asemena, clasa `ScrollPane` oferă posibilitatea de a se afişa dimensiunea minimă, dimensiunea maximă precum şi dimensiunea curentă a componentelor pe care le conține.

[42] Aceste proprietăți pot fi obținute, însă nu pot fi specificate pentru a determina valorile selectate, respectiv focusate în cadrul listei.

Tabelele JavaFX au capacitatea de a sorta datele[43] şi de a redimensiona coloanele în funcție de conținut. În cazul în care se doreşte editarea conținutului unor tabele, se va folosi metoda `setEditable(true)`. Structura unui tabel presupune existența unor coloane (obiecte de tip `TableColumn`[44]) care se adaugă la obiectul de tip `TableView`.

idcarte	titlu	descriere	ideditura	anaparitie	editie	idcolectie	iddomeniu	stoc
1	De ce vedem filme	-	23	1882	6	79	80	490
2	Trickster	-	86	1904	7	69	27	449
3	Lolita	-	18	1945	10	40	57	522
4	Demonii Vantului	-	22	1885	2	13	46	189
5	Trickster	-	40	1922	1	73	85	247
6	Cinema	-	1	1877	5	1	1	676
7	Iubirile croitoresei	-	23	1855	7	37	6	323
8	De ce vedem filme	-	21	1898	6	64	93	679
9	De ce vedem filme	-	39	1830	10	86	51	714
10	Forta politica a femeilor	-	64	1810	8	21	50	578
11	Gandirea Salbatica	-	40	1915	2	43	83	246
12	8 metode eficiente pentru educarea copiilor	-	12	1853	1	61	52	87
13	Iubirile croitoresei	-	90	1887	9	94	75	169
14	8 metode eficiente pentru educarea copiilor	-	47	1821	6	99	49	73

Figura 6 – Obiect de tip `TableView` conținând tabela „cărți" din aplicația BookStore

Clasa `TreeView` din pachetul `javafx.scene.control` permite vizualizarea de conținuturi care respectă o structură ierarhică în care elementul cel mai de sus este numit rădăcină, iar elementele cele mai de jos sunt numite frunze. Construirea unui astfel de obiect implică definirea mai multor obiecte `TreeItem`[45] între care se stabilesc relații de tip părinte-fiu. Obiectul `TreeView` va fi creat având ca parametru obiectul `TreeItem` rădăcină (alternativ, se poate utiliza metoda `getRoot`). Un obiect `TreeView` poate conține elemente cu tipuri diferite. Clasa `TreeView` este extinsă din `Node`, în timp ce `TreeItem` nu, motiv pentru care asupra obiectelor de acest tip nu pot fi aplicate efecte vizuale. În acest sens, trebuie definit un mecanism de tip „fabrică" pentru a genera obiecte de tip `TreeCell` al căror comportament poate fi modificat în funcție de necesități.

Obiectele de tip `ComboBox` sunt folosite atunci când este necesară selectarea unei opțiuni dintr-o listă care depăşeşte o anumită limită, pentru că oferă posibilitatea vizualizării folosind elemente derulante. Elementele conținute (obiect de tip listă observabilă) sunt specificate fie la construirea obiectului, fie folosind metoda `addAll()` aplicată unui obiect obținut din `getItems()`. Selecția poate fi specificată prin metoda `setValue()`[46], sau poate fi obținută prin metoda `getValue()`. Numărul de opțiuni vizibile din cadrul listei va fi modificat folosind metoda `setVisibleRowCount()`, primind ca parametru numărul de elemente care vor fi afişate. Ca şi alte controale JavaFX, elementele de tip `ComboBox` pot fi editate (`setEditable(true)`), putându-se specifica şi un text explicativ (`setPromptText`).

[43] Sortarea obiectelor din cadrul unei coloane se face printr-un click al mouse-ului. În funcție de numărul de click-uri al mouse-ului, elementele vor fi sortate crescător (1), descrescător (2) sau deloc (3). Programatorul poate indica din cod preferințele pentru sortarea conținutului, folosind metoda `setSortType`, care primeşte ca parametru o constantă din clasa `TableColumn.SortType` (având valorile `ASCENDING` / `DESCENDING`). Coloanele care vor fi sortate pot fi indicate prin apăsarea tastei Shift înainte de selectarea lor, fie prin metoda `TableView.sortOrder` având ca parametru lista observabilă a coloanelor.

[44] Pentru un obiect de tip `TableColumn` se poate utiliza metoda `setVisible(false)` dacă nu se doreşte vizualizarea conținutului său.

[45] Obiectele `TreeItem` pot fi expandate (vizualizându-se conținutul lor) sau nu, în funcție de parametrul metodei `setExpanded`. Implicit, conținutul obiectelor `TreeItem` nu poate fi vizualizat.

[46] Metoda `setValue()` modifică şi obiectul asociat din proprietatea `selectionModel`.

Elementul `Separator` este utilizat doar pentru a distinge între elemente, fără a produce vreo acțiune. El poate fi însa stilizat, i se pot adăuga efecte vizuale şi poate fi animat. Un `Separator` poate fi orizontal sau vertical, orientarea sa modificându-se prin metoda `setOrientation`. Dimensiunea unui astfel de obiect poate fi precizată în cadrul metodei `setMaxWidth`, în timp ce poziționarea sa (orizontală sau verticală) se face cu `setValignement`, respectiv `setHalignement`.

Un control de tip `Slider` conține o pistă şi un cursor care poate fi poziționat. Totodată, poate include marcaje ce pot avea etichete asociate, indicând diferite valori numerice din intervalul respectiv.

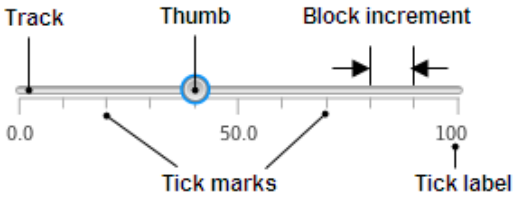


Figura 7 – Elementele constitutive ale unui obiect de tip `Slider`

Metodele pe care le pune la dispoziție această clasă sunt asemănătoare cu cele din clasa `ScrollPane`, având aceeaşi semnificație (`setMin`, `setMax`, `setValue`, `setShowTickLabels`, `setShowTickMarks`[47], `setMajorTickUnit`, `setMinorTickCount`, `setBlockIncrement`).

Prin clasele `ProgressIndicator` şi `ProgressBar`[48] se indică utilizatorului faptul că o sarcină este în curs de execuție, precizându-se şi care este cantitatea din cadrul acesteia care a fost terminată deja. Obiectele `ProgressIndicator` sunt de forma unui grafic circular al cărui conținut se umple pe măsură ce procentul este mai mare, în timp ce obiectele `ProgressBar` au forma unei bare. Constructorul primeşte ca parametru procentul în care sarcina a fost completată[49] (alternativ se poate folosi `setProgress`), existând posibilitatea ca obiectele să fie create fără parametru, în cazul în care nu se poate evalua dimensiunea sarcinii în cauză[50]. Clasele mai pun la dispoziția programatorilor metodele `build()` şi `progress()`.

Clasa `Hyperlink`, derivată din `Labeled` (din care moşteneşte metodele `setText` şi `setGraphic`) este folosită spre a formata texte care au semnificația legăturilor Internet. Starea unei legături (vizitată / nevizitată) poate fi marcată prin metoda setVisited. La aceesarea unui astfel de obiect, locația poate fi vizualizată printr-un obiect `WebEngine` (obținut ca `new WebView().getEngine()`) pentru care se apelează metoda `load`.

Obiectele `Tooltip` sunt folosite pentru texte explicative ce pot fi asociate oricăror controale JavaFX prin apelul metodei `setTooltip`. Starea sa poate fi activ şi vizibil, de regulă existând o întârziere între cele două momente (plasarea mouse-ului asupra obiectului şi afişarea propriu-zisă). Fiind derivată din `Labeled`, i se poate asocia nu numai un text, ci şi o imagine.

---

[47] În cazul în care se doreşte alinierea cursorului cu marcajele din cadrul pistei, se poate folosi metoda `setSnapToTicks`.

[48] `ProgressBar` reprezintă o subclasă directă a `ProgressIndicator`.

[49] Parametrul va avea o valoare subunitară.

[50] Într-o astfel de situație, constructorul poate primi ca parametru şi o valoare negativă. Metoda `isIndeterminate` se foloseşte pentru a determina dacă progresul este într-o stare nedeterminată sau nu.

Controlul `HTMLEditor` este un editor de texte cu numeroase funcționalități pentru redactarea documentelor HTML5:

- formatare de text (bold, italic, underline, strike though);
- setări cu privire la paragraf (formatare, tip şi dimensiune a caracterelor);
- culori de fundal;
- indentare de text;
- liste numerotate şi nenumerotate;
- aliniere de text;
- folosirea unei rigle orizontale;
- facilități copy/paste pentru fragmente de text.

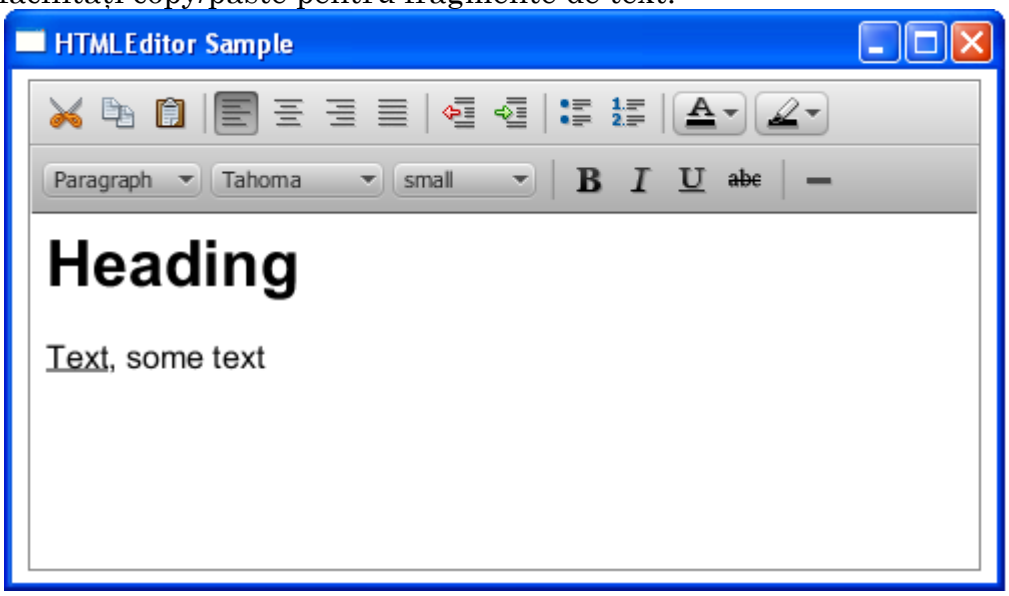


Figura 8 – Un control de tip `HTMLEditor`

În cadrul editorului, conținutul este afişat sub forma unui şir de caractere reprezentând codul HTML asociat. Constructorul său nu primeşte parametri, dar poate fi specificată dimensiunea prin metodele `setPrefWidth` / `setPrefHeight`. Totodată conținutul său (o pagină Internet dată prin codul HTML) va fi specificat prin metoda `setHtmlText`. Acesta poate fi obținut folosind metoda „pereche" `getHtmlText`.

Obiectele de tip `TitledPane` reprezintă panouri care au asociat şi un titlu, putând fi deschise şi închise, încapsulând orice obiect de tip `Node` (controale, grupuri de elemente din cadrul unui container). Gruparea obiectelor `TitledPane` se face prin clasa `Accordion`, acestea putând fi astfel afişate împreună. Obiectele `TitledPane` pot fi construite primind ca parametrii textul / obiectul de tip `Node`, acestea putând fi specificate ulterior prin metodele `setText`, respectiv `setContent`[51]. Implicit, orice obiect TitledPane poate fi închis şi deschis (proprietate ce poate fi specificată prin metoda `setCollapsible`), acțiunea realizându-se în mod animat (metoda `setAnimated` indică acest comportament).

Un obiect `Accordion` conține mai multe obiecte `TitledPane` cu proprietatea că unul singur dintre acestea poate fi vizualizat la un moment dat[52].

[51] Se recomandă ca înălțimea (minimă / maximă sau preferată) a unui obiect de tipul `TitledPane` să nu fie specificată întrucât acest fapt poate conduce la evenimente nedorite în cazul în care acesta este expandat.

[52] Obiectul `TitledPane` vizualizat în mod curent este accesat prin `{set/get}ExpandedPane`.

În JavaFX pot fi definite mai multe tipuri de meniuri[53] (`MenuBar`, `MenuItem` – `CheckMenuItem`, `RadioMenuItem`, `CustomMenuItem`, `ContextMenu`). Se construieşte obiectul `MenuBar`, se definesc categoriile care sunt populate cu elemente `MenuItem`, corespunzând unei opțiuni acționabile. Obiectele `Menu` sunt utilizate pentru implementarea unui submeniu `RadioButtonItem` pentru selecții mutual exclusive iar `CheckMenuItem` pentru elemente a căror stare are o valoare binară. Separația între aceste elemente se face prin intermediul obiectelor `SeparatorMenuItem`. Obiectele `MenuBar` ocupă (de regulă) partea de sus a scenei. Dacă spațiul nu permite acest lucru se pot folosi meniuri contextuale (`ContextMenu`).

```
applicationMenu = new MenuBar();
for (int currentIndex1=0;
     currentIndex1 < Constants.MENU_STRUCTURE.length;
     currentIndex1++) {
 Menu menu = new Menu(Constants.MENU_STRUCTURE[currentIndex1][0]);
 for (int currentIndex2 = 1;
      currentIndex2 < Constants.MENU_STRUCTURE[currentIndex1].length;
      currentIndex2++) {
  MenuItem menuItem =
  new MenuItem(Constants.MENU_STRUCTURE[currentIndex1][currentIndex2]);
  menuItem.addEventHandler(EventType.ROOT, this);
  menu.getItems().add(menuItem);
 }
 applicationMenu.getMenus().add(menu);
}
```

În exemplul dat, se poate observa ierarhia `MenuBar` → `Menu` → `MenuItem`.

Pentru un obiect `MenuItem`, se poate asocia o combinație de taste având acelaşi efect ca şi accesarea propriu-zisă a acestuia (folosind mouse-ul), apelând metoda `setAccelerator`, primind un parametru obținut din clasa `KeyCombination`.

Obiectele `CheckMenuItem` pot avea starea selectat / deselectat, proprietate accesibilă din proprietatea `selectedProperty`.

Obiectele `RadioMenuItem` sunt de obicei asociate unui obiect `ToggleGroup` care implică selecția lor mutual exclusivă.

Obiectele `ContextMenu` sunt afişate atunci când este apelată metoda `show` asociată, de regulă la accesarea unui control căruia i-a fost alocat meniul.

Controlul `ColorPicker` este un element de interfață cu utilizatorul care permite selecția unei culori dintr-o anumită gamă dar şi specificarea acesteia[54] prin parametrii RGB (Red/Green/Blue) sau HSB (Hue/Saturation/Brightness). Obiectul conține o listă pentru selecția culorii, o paletă de culori precum şi fereastra de dialog pentru specificarea parametrilor acesteia. Obiectul se poate crea fără parametri sau poate fi indicată o anumită culoare, considerată curentă (aceasta poate fi gestionată şi prin intermediul metodelor `{get/set}Value`). Totodată, culorile definite de utilizator pot fi obținute folosindu-se metoda `getCustomColors` (al cărui rezultat este un obiect de tipul `ObservableList<Color>`).

[53] Un meniu se defineşte ca o listă de elemente acționabile ce sunt afişate la cererea utilizatorului (atunci când un meniu este vizibil, utilizatorul poate selecta un element al meniului, urmând ca după realizarea selecției în cauză acesta să dispară). Acesta reprezintă şi un mod de a economisi spațiul disponibil, plasându-se în cadrul meniurilor funcționalitatea care nu trebuie să fie vizibilă permanent.

[54] Utilizatorii îşi pot defini o culoare şi prin mişcarea mouse-ului asupra suprafeței de culoare şi asupra barei de culori, parametrii RGB/HSB fiind modificați în mod automat. De asemenea, culoarea poate fi modificată şi indicând valoarea acesteia în format web (#, urmat de 6 cifre hexa).

Clasa `Pagination` oferă posibilitatea de a naviga între mai multe pagini corespunzând unui conținut împărțit în mai multe secțiuni. Controlul este format din conținutul propriu-zis al paginii (redat în funcție de logica aplicației), dar şi din zona de navigare[55]. Constructorul unui obiect de acest tip poate fi apelat şi fără parametri (caz in care numărul total de pagini are valoarea `INDETERMINATE`), poate primi 1 parametru indicând numărul total de pagini sau 2 parametrii, specificându-se şi numărul paginii selectate[56]. Nu se poate adăuga conținut asociat unei pagini decât prin intermediul unei „fabrici" de pagini, folosindu-se metoda `setPageFactory` care va folosi o metodă de tip callback[57].

```
Pagination pagination = new Pagination (5, 0);
Pagination.setPageFactory(new Callback<Integer, Node>() {
      @Override
      public Node call(Integer currentPage) {
      ...
      }
});
```

În condițiile în care nu pot fi afişate în zona de navigare butoane corespunzând tuturor paginilor, se poate utiliza metoda `setMaxPageIndicatorCount` care limitează dimensiunea numărului de legături vizibile către pagini.

În aplicațiile JavaFX componentele text pot fi create ca instanțe ale clasei `javafx.scene.text.Text` (derivată din clasa `Node`)[58], respectiv `javafx.scene.text.TextBuilder`. În cazul obiectelor `Text`, la creare se poate specifica (sau nu) textul respectiv, precum şi poziția unde se doreşte a fi afişat. Pentru aceste obiecte se pot utiliza şi metodele `setText`, `setFont`[59] şi `setColor`. Atunci când se folosesc obiecte de tip `TextBuilder` se foloseşte metoda statică `create`, apelându-se succesiv metoda `text` (pentru a specifica şirul de caractere respectiv) şi apoi metoda `build`.

Efectele care se pot crea asupra textelor[60] sunt `PerspectiveTransform` (indicându-se cele 4 puncte ale formei ce defineşte transformarea de perspectivă ca perechi (x,y)), `GaussianBlur` pentru estomparea textului, `DropShadow` pentru umbre exterioare, `InnerShadow` pentru umbre interioare, `Reflection` pentru reflectarea conținutului respectiv, indicându-se şi o anumită fracție (`setFraction`)

[55] În zona de navigare se găsesc butoane pentru deplasarea înainte şi înapoi, butoane pentru fiecare pagină în parte (cel selectat fiind corespunzător paginii curente). Totodată, pagina curentă este specificată şi printr-o etichetă având forma „pagina curenta / număr total de pagini". Navigarea se face fie prin accesarea butoanelor înainte şi înapoi, fie prin accesarea butonului corespunzător paginii în cauză.

[56] În cazul specificării unei pagini curente, numerotarea începe întotdeauna de la 0. Alternativ, pot fi folosite metodele `setPageCount`, respectiv `setCurrentPageIndex`.

[57] Metoda de tip callback va fi apelată de fiecare dată când este selectată o pagină.

[58] În acest mod se pot aplica efecte sau transformări şi se pot crea animații folosind obiectul text respectiv. De asemenea, având în vedere că `Node` extinde `Shape`, se poate specifica şi o culoare pentru fundalul textului, respectiv se poate aplica un efect de îngroşare pentru forma asociată lui.

[59] Parametrul unei astfel de metode se obține prin metoda statică `font` a clasei cu acelaşi nume, specificându-se numele tipului de caractere (ca String) şi dimensiunea acestuia. De asemenea, proprietățile de tip bold / italic pot fi specificate ca parametri ai aceleiaşi metode, utilizându-se constantele `FontWeight.BOLD` respectiv `FontPosture.ITALIC`. Se pot utiliza şi tipuri de caractere definite de utilizator (conținute în fişiere de tip .ttf), acestea putând fi încărcate prin `loadFont`.

[60] Aplicarea unui efect se face prin metoda `setEffect`, care primeşte ca parametru obiect de tipul transformării respective.

Combinarea de efecte se face folosind clasa `Blend` (modul fiind în acest caz `BlendMode.MULTIPLY`), efectele fiind specificate ca parametri ai metodelor `setBottomInput`, respectiv `setTopInput`[61].

5. Tratarea evenimentelor asociate controalelor JavaFX

Evenimentele reprezintă mecanisme utilizate pentru a transmite aplicației acțiunile realizate de utilizator, astfel încât aceasta să reacționeze corespunzător. În JavaFX, un eveniment e o instanță a clasei `javafx.event.Event` sau a oricărei subclase a acesteia[62].

Tabelul 2 – Proprietățile unui eveniment

Proprietate	Descriere
tip eveniment	tipul evenimentului care s-a produs
sursă	originea evenimentului, raportat la locația evenimentului în lanțul de propagare[63]
țintă	nodul asupra căruia a avut loc acțiunea şi nodul frunză în lanțul de propagare[64]

Un **tip de eveniment** este o instanță a clasei `EventType`, acestea fiind organizate ierarhic, pe nivelul cel mai înalt aflându-se `Event.ROOT` echivalent cu `Event.ANY`[65].

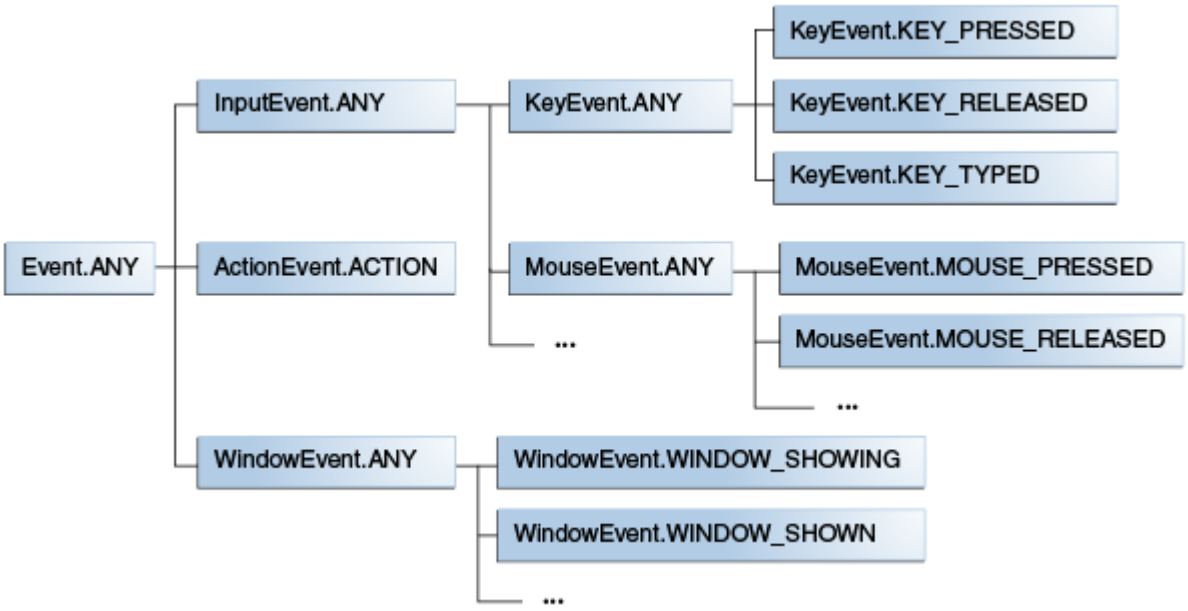


Figura 9 – Ierarhia tipurilor de evenimente [5]

Sursa unui eveniment desemnează nodul care are asociat mecanismul de tratare a evenimentului, aflându-se cel mai aproape de nodul propriu-zis (asupra căruia s-a exercitat acțiunea respectivă) în ierarhia de obiecte a scenei.

---

[61] Atunci când se dorește combinare a mai mult de 2 efecte, combinarea se va face între un efect propriu-zis şi alt obiect de tip `Blend` care conține la rândul său o altă combinație.
[62] Pe lângă tipurile predefinite de evenimente (`DragEvent`, `KeyEvent`, `MouseEvent`, `ScrollEvent`), programatorul își poate defini propriile tipuri de evenimente (care vor extinde clasa `Event`).
[63] Sursa se modifică pe măsură ce evenimentul este transmis de-a lungul lanțului de propagare.
[64] Deşi ținta nu se schimbă, dacă un filtru pentru evenimente consumă evenimentul în procesul de tratare a avenimentului, ținta nu va primi evenimentul.
[65] Pe fiecare nivel, subtipul `ANY` este folosit pentru a desemna orice tip de eveniment din clasă. Mecanismul este foarte util în cazul în care se folosesc filtre pentru evenimente, putându-se trata fie doar anumite tipuri de evenimente, fie toate tipurile de evenimente specifice clasei respective.

Ținta unui eveniment este instanța clasei `EventTarget` sau a oricărei subclase a acesteia. Lanțul de transmitere al evenimentului (ruta pe care evenimentul o urmează pentru a ajunge la țintă) este realizată prin metoda `buildEventDispatchChain`[66].

Prin **lanț de propagare a unui eveniment** se înțelege un proces format din următoarele etape: selecția țintei, stabilirea parcursului urmat de eveniment, capturarea propriu-zisă evenimentului şi întoarcerea sa la nodul rădăcină.

În etapa de *selecția țintei*, se determină care este nodul țintă, în funcție de următoarele reguli:

- în evenimente legate de tastatură, nodul țintă este cel ce deține controlul la momentul respectiv;
- în evenimente legate de mouse, nodul țintă este cel de la locația cursorului;
- în evenimente ce implică gestică în cazul dispozitivelor cu ecran tactil, nodul țintă este cel situat în centrul acțiunilor utilizatorului, la începutul gestului respeciv[67];
- în evenimente de tip derulare (*eng.* swipe) într-un dispozitiv cu ecran tactil nodul țintă este cel situat în centrul acțiunilor utilizatorului, considerându-se întregul parcurs al acestuia;
- în evenimente de tip apăsare pentru dispozitive cu ecran tactil, nodul țintă pentru fiecare punct este cel de la locația primei apăsări[68].

În cazul în care la locația evenimentului se află mai multe noduri, ținta este cea de la nivelul cel mai înalt. De asemenea, când se produce un eveniment cum ar fi o apăsare la nivel de tastatură sau de mouse, evenimentele succesive (până ce tasta sau butonul respectiv sunt eliberate) vor fi legate de nodul țintă inițial.

Pentru *stabilirea parcursului urmat de eveniment*, în principiu se foloseşte lanțul de propagare a evenimentului dat de metoda `buildEventDispatchChain()` de la nivelul nodului țintă selectat (aceasta va conține în mod obligatoriu obiectele `Stage` şi `Scene` care conțin nodul respectiv şi apoi calea propriu-zisă către nodul respectiv în graful de scene[69]).

În etapa de *capturare propriu-zisă*, evenimentul în cauză este transmis de la nodul rădăcină către nodul țintă prin lanțul de propagare a evenimentului. Fiecare nod din acest lanț de propagare al evenimentului care are implementată o metodă pentru respectivul tip de eveniment îl va gestiona, ulterior transmițându-l mai departe. Dacă pe parcurs nu există un filtru pentru eveniment care să îl consume, acesta va ajunge la nodul țintă într-un final.

---

[66] Clasele `Window`, `Scene` şi `Node` implementează interfața `EventTarget` şi toate subclasele lor moştenesc această proprietate, ceea ce înseamnă că pentru majoritatea obiectelor nu va fi necesar să se stabilească lanțul de transmitere a evenimentului, implementându-se doar modul de reacție la evenimentul respectiv. Pentru controalele definite de utilizator, în cazul în care nu sunt subclase ale `Window`, `Scene` şi `Node`, pentru acestea va trebui să se implementeze interfața `EventTarget`, pentru ca acesta să gestioneze acțiunile utilizatorului.

[67] În cazul simulării unor evenimente ce implică gestică (de către un dispozitiv ce nu conține ecran tactil), nodul țintă este cel de la locația cursorului.

[68] De asemenea, se poate specifica un alt nod țintă folosind metodele `grab[(node)]` sau `ungrab` pentru un anumit punct într-un filtru de evenimente sau mecanism de tratare a evenimentelor.

[69] Totuşi, ruta poate fi modificată după cum filtrele de evenimente respectiv metodele de tratare pentru evenimente de pe parcursul urmat de eveniment îl procesează (spre exemplu, dacă evenimentul este „consumat” înainte de a ajunge la nodul țintă).

După ce evenimentul a ajuns la nodul țintă şi / sau a fost procesat de către toate filtrele, acesta se va *întoarce la nodul rădăcină*, urmărind parcursul dat de lanțul de propagare, însă în sens invers. Dacă vreun nod în lanțul de propagare are definită o metodă de tratare, aceasta este apelată, transmițând evenimentul mai departe după ce se termină. Evenimentul va fi transmis într-un final şi nodului rădăcină doar dacă metodele de tratare a acestuia nu îl consumă.

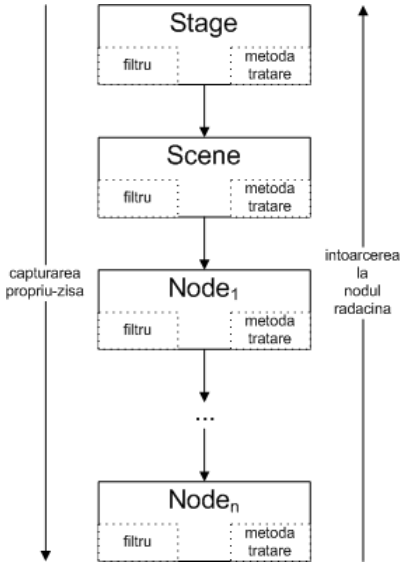


Figura 10 – Mecanismul de filtrare şi tratare al evenimentelor în JavaFX

În JavaFX, gestiunea evenimentelor se face prin filtre şi metode de tratare, acestea fiind implementări ale interfeței `EventHandler`. Atunci când se vrea ca aplicația să fie notificată de apariția unui eveniment, trebuie asociat fie un filtru, fie o metodă de tratare specifică evenimentului respectiv[70].

Un *filtru* este executat în etapa de capturare propriu-zisă. La nivelul unui nod părinte, pot fi implementate filtre pentru mai multe noduri fiu şi în caz de necesitate, evenimentul poate fi consumat spre a preveni ca acesta să mai ajungă la nodul țintă. Un nod poate avea asociate mai mult de un singur filtru. Ordinea în care acestea sunt executate ține de ierarhia tipurilor de evenimente[71].

O *metodă de tratare* e executată în etapa de întoarcere la nodul rădăcină. Dacă metoda de tratare de la nivelul nodului fiu nu consumă evenimentul, atunci nodul părinte îl poate gestiona ulterior. Şi în cazul metodelor de tratare se aplică aceeaşi regulă din cazul filtrelor cu privire la numărul de evenimente pe care acestea îl pot gestiona precum şi a ordinii în care acestea sunt executate[72].

Un eveniment poate fi consumat de un filtru sau de o metodă de tratare oricând în lanțul de propagare al evenimentelor prin metoda `consume` care anunță faptul că procesarea evenimentului s-a încheiat. Consumarea unui eveniment de un filtru împiedică nodurile fii să îl primească la fel cum consumarea sa de către o metodă de tratare împiedică nodurile părinte să îl primească. Nu este afectată însă gestiunea altor tipuri de evenimente din cadrul nodului respectiv.

---

[70] Diferența dintre un filtru şi o metodă de tratare constă în momentul la care fiecare dintre acestea sunt executate.

[71] Filtrele asociate unui tip de eveniment mai specific sunt executate înainte de filtrele asociate unui tip de eveniment mai generic. Nu se specifică însă ordinea în care sunt executate filtrele corespunzătoare unor tipuri de evenimente aflate pe acelaşi nivel.

[72] Evenimentele specificate prin metode de oportunitate (*eng.* convenience methods) se execută ultimele.

Unele clase JavaFX definesc proprietăți pentru metodele de tratare, oferind un mecanism de a specifica un comportament în cazul în care se produc anumite evenimente. Acestea poartă numele de *mecanisme de oportunitate* pentru asocierea unor metode de tratare. Acestea sunt definite în clasa `Node`, fiind disponibile pentru toate subclasele lor. Alte clase işi au definite propriile mecanisme de oportunitate.

Tabelul 3 – Clase cu mecanisme de oportunitate pentru tratarea evenimentelor [5]

Acțiunea Utilizatorului	Tip de Eveniment	Clasa
Se apasă o tastă de pe tastatură.	`KeyEvent`	`Node, Scene`
Mouse-ul este mişcat sau se apasă un buton de pe mouse.	`MouseEvent`	`Node, Scene`
Se produce o secvență de acțiuni care implică apăsarea unui buton de pe mouse, mişcarea mouse-ului urmată de eliberarea butonului.	`MouseDragEvent`	`Node, Scene`
Se transmite conținut printr-o metodă alternativă de introducere a caracterelor (limbă ce utilizează alte tipuri de caractere)	`InputMethodEvent`	`Node, Scene`
Un obiect este mutat între locații prin selectarea, transportarea şi deselectarea sa.	`DragEvent`	`Node, Scene`
Obiectul este derulat.	`ScrollEvent`	`Node, Scene`
Se produce un gest de tip rotație asupra obiectului.	`RotateEvent`	`Node, Scene`
Se produce un eveniment de tip derulare asupra unui obiect.	`SwipeEvent`	`Node, Scene`
Pe ecranul tactil, este atinsă locația la care se află un obiect.	`TouchEvent`	`Node, Scene`
Se produce un gest ce implică modificarea rezoluției asupra obiectului respectiv.	`ZoomEvent`	`Node, Scene`
Este solicitată vizualizarea meniului contextual.	`ContextMenuEvent`	`Node, Scene`
Butonul este apăsat. Lista derulantă este vizibilă sau este ascunsă. Elementul unui meniu este selectat.	`ActionEvent`	`ButtonBase, ComboBoxBase, ContextMenu, MenuItem, TextField`

Se editează elementele dintr-un obiect de tip listă, tabel sau arbore.	`ListView.EditEvent`
	`TableColumn.CellEditEvent`
	`TreeView.EditEvent`
		`ListView`
		`TableColumn`
		`TreeView`

Obiectul de redare a conținutului multimedia întâlneşte o eroare.	`MediaErrorEvent`	`MediaView`
Meniul este vizibil sau ascuns.	`Event`	`Menu`
O fereastră pop-up este ascunsă.	`Event`	`PopupWindow`
Panoul este închis sau selectat.	`Event`	`Tab`
Fereastra este închisă, vizibilă sau ascunsă.	`WindowEvent`	`Window`

O metodă de tratare e asociată nodului prin mecanismul de oportunitate folosind următoarea sintaxă:

```
setOnEvent-type(EventHandler<? super event-class> value)
```

unde

`event-type` este tipul evenimentului pe care îl gestionează metoda de tratare

`event-class` este clasa care defineşte tipul de eveniment[73].

Totodată, se poate implementa metoda de tratare prin definirea acesteia într-o clasă anonimă în cadrul mecanismului de oportunitate, specificându-se comportamentul în cazul producerii evenimentului în contextul funcției `handle`.

```
tableContent.setOnMouseClicked(new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent event) {
    // ...
    }
});
```

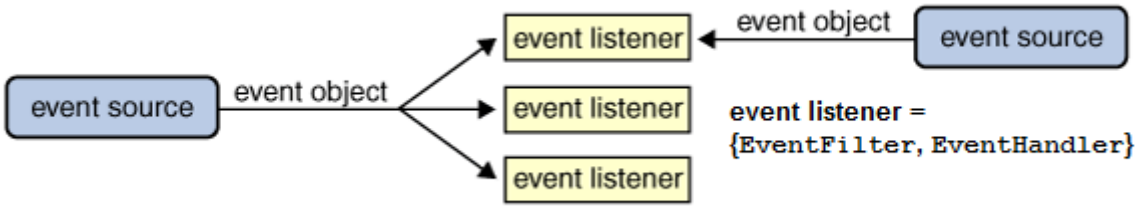


Figura 11 – Modelul observer-listener pe cazul general

Un filtru de eveniment permite gestiunea evenimentului în timpul etapei de capturare propriu-zisă. Un nod poate avea asociate unul sau mai multe filtre pentru gestiunea unui eveniment. De asemenea un filtru poate fi utilizat pentru unul sau mai multe noduri, pentru unul sau mai multe tipuri de evenimente. Totodată, nodurile părinte pot oferi o procesare de nivel înalt a evenimentului pentru nodurile fiu, interceptându-le astfel ca acestea să nu mai ajungă la țintă. Pentru a gestiona un eveniment în timpul etapei de capturare propriu-zisă, nodul trebuie sa asocieze un filtru de eveniment, care implementează interfața `EventHandler`. Metoda `handle` conține codul executat la momentul în care evenimentul ajunge la nodul care are filtrul asociat.

Asocierea unui filtru de eveniment pentru un nod se face prin metoda `addEventFilter`, primind ca parametri tipul de eveniment şi filtrul propriu-zis. Eliminarea unui filtru de eveniment corespunzător unui nod se face prin metoda `removeEventFilter`, primind aceeaşi parametri.

```
node.addEventFilter(MouseEvent.MOUSE_CLICKED,
                    new EventHandler<MouseEvent>() {
                        public void handle(MouseEvent) { ... };
                    });
EventHandler filter = new EventHandler(<InputEvent>() {
    public void handle(InputEvent event) {
        // ...
        event.consume();
    }
node1.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);
node2.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);
node.addEventFilter(KeyEvent.KEY_PRESSED, filter);
node.removeEventFilter(KeyEvent.KEY_PRESSED, filter);
```

[73] Sintaxa `<? super event-class>` indică faptul că metoda acceptă o metodă de tratare fie pentru clasa care defineşte tipul de eveniment (`event-class`), fie pentru una dintre superclasele acesteia.

O metodă de tratare a unui eveniment permite gestiunea evenimentului în timpul etapei de întoarcere la nodul rădăcină. Un nod poate avea asociată una sau mai multe metode pentru tratare a unui eveniment. De asemenea, o metodă pentru tratarea unui eveniment poate fi folosită pentru unul sau mai multe noduri, respectiv unul sau mai multe tipuri de eveniment. În cazul când o metodă de tratare a unui eveniment de la nivelul nodului fiu nu consumă evenimentul, acesta va fi gestionat şi de metoda de tratare de la nivelul nodului părinte, oferind un mecanism de gestiune mai general caracteristic tuturor nodurilor fiu. Pentru a gestiona un eveniment în timpul etapei de întoarcere la nodul rădăcină, nodul trebuie sa asocieze o metodă de tratare, care implementează interfața `EventHandler`. Metoda `handle` conține codul executat la momentul în care evenimentul ajunge la nodul care are metoda de tratare asociată.

Asocierea unei metode de tratare pentru un nod se face prin metoda `addEventHandler`, primind ca parametri tipul de eveniment şi filtrul propriu-zis. Eliminarea unei metode de tratare corespunzătoare unui nod se face prin metoda `removeEventHandler`, primind aceeaşi parametri.

```
node.addEventHandler(DragEvent.DRAG_ENTERED,
                     new EventHandler<DragEvent>() {
                         public void handle(DragEvent) { ... };
                     });
EventHandler handler = new EventHandler(<InputEvent>() {
    public void handle(InputEvent event) {
        // ...
        event.consume();
    }

node1.addEventHandler(DragEvent.DRAG_EXITED, handler);
node2.addEventHandler(DragEvent.DRAG_EXITED, handler);
node.addEventHandler(MouseEvent.MOUSE_DRAGGED, handler);
node.removeEventHandler(DragEvent.DRAG_EXITED, handler);
```

În cazul în care s-au folosit mecanisme de oportunitate, eliminarea metodei de tratare a evenimentului se face prin specificarea unui parametru cu valoarea null pentru nodul în cauză.

```
node.setOnMouseDragged(null)
```

6. Utilizarea FXML în JavaFX

FXML reprezintă un limbaj bazat pe XML care oferă structura necesară pentru dezvoltarea unei interfețe cu utilizatorul, separând-o de logica aplicației. Deşi nu urmăreşte o schemă, FXML are o structură de bază predefinită. Astfel, cele mai multe clase JavaFX pot fi utilizate ca elemente şi cele mai multe proprietăți (ale unor componente de tip JavaBeans) pot fi folosite ca atribute. Dintr-o perspectivă a arhitecturii MVC (Model View Controller), fişierul FXML care conține descrierea interfeței cu utilizatorul reprezintă partea de vizualizare. Controlul este realizat prin clasele Java[74], iar modelul va fi reprezentat prin obiecte domeniu (definite în context Java) ce se asociază vizualizării prin control. FXML este extrem de util în special pentru interfețele cu utilizatorul ce conțin grafuri de scene complexe, de dimensiuni mari, formulare, animații complicate, precum şi pentru interfețe statice (formulare, controale, tabele). Pot fi definite şi interfețe dinamice prin utilizarea de scripturi.

[74] Aceasta poate implementa în mod opțional clasa `Initializable`, care este declarată drept control pentru fişierul FXML.

Printre beneficiile FXML se numără următoarele [6]:

- are o sintaxă similară cu XML, punând la dispoziția diferiților dezvoltatori un mecanism familiar pentru a construi interfețe cu utilizatorul;
- graful de scene este mai transparent în FXML, facilitând atât construirea cât şi menținerea interfeței cu utilizatorul;
- FXML nu este un limbaj compilat, astfel încât proiectul nu trebuie reconstruit atunci când se produc modificări ca acestea să fie vizibile;
- conținutul unui fişier FXML poate fi localizat pe măsură ce este parcurs[75];
- FXML este compatibil cu orice limbaj JVM (Java Virtual Machine) ca: Java, Scala,Clojure;
- FXML nu este limitat doar la partea de vizualizare a arhitecturii MVC, ci se pot construi servicii, sarcini sau obiecte domeniu, existând şi o integrare cu limbaje pentru scripturi (ca JavaScript).

În versiunea 2.2 a JavaFX au fost introduse câteva facilități:

- eticheta `<fx:constant>` înlesneşte procesul de căutare a constantelor `java.lang.Double.NEGATIVE_INFINITY` `<Double fx:constant="NEGATIVE_INFINITY"/>`
- accesul la sub-clasele care asigură controlul în FXML a fost îmbunătățit – pentru fiecare nod se pot defini clase în cadrul cărora sunt tratate evenimente; în cazul în care acesta conține la rândul său alte noduri, atunci când acestea sunt create se va construi şi sub-clasele care asigură controlul (în cadrul metodei `initialize`).
- inițializarea claselor care asigură controlul se face prin mecanismul de reflecție, instanța clasei `FXMLLoader` identificând automat metoda `initialize`[76] pe care o şi apelează;
- uşurința în realizarea de noi controale FXML – prin intermediul metodelor `setRoot` şi `setController`, codul sursă care le apelează poate stabili valori pentru nodul rădăcina, respectiv pentru obiectul care se ocupă de control astfel încât aceste operații nu mai trebuie realizate de către FXMLLoader; se pot dezvolta astfel controale care pot fi reutilizate.

În exemplul de mai jos, este creată o interfață cu utilizatorul pentru procesul de autentificare, specific celor mai multe dintre aplicațiile integrate pentru întreprinderi:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>

<AnchorPane fx:id="AnchorPane" prefHeight="150.0" prefWidth="400.0"
xmlns:fx="http://javafx.com/fxml" fx:controller="BookStore">
  <children>
    <GridPane layoutX="15.0" layoutY="15.0">
```

[75] Acest lucru nu se întâmplă în cazul codului sursă dezvoltat în limbajul de programare Java unde conținutul fiecărui element trebuie modificat în mod manual (se obține o referință către el după care se apelează metoda de stabilire a proprietății corespunzătoare).
[76] În cazul în care nu este publică, pentru această metodă trebuie folosită adnotarea `@FXML`.

```
      <children>
        <Label text="Nume Utilizator:" GridPane.columnIndex="0"
GridPane.rowIndex="0" />
        <Label text="Parola:" GridPane.columnIndex="0"
GridPane.rowIndex="1" />
        <TextField fx:id="campTextNumeUtilizator" prefWidth="200.0"
GridPane.columnIndex="1" GridPane.rowIndex="0" />
        <PasswordField fx:id="campTextParola" prefWidth="200.0"
GridPane.columnIndex="1" GridPane.rowIndex="1" />
        <HBox prefHeight="100.0" prefWidth="200.0" spacing="5.0"
GridPane.columnIndex="1" GridPane.rowIndex="2">
          <children>
            <Button fx:id="butonAcceptare" mnemonicParsing="false"
onAction="#handleButonAcceptareAction" text="Acceptare" />
            <Button fx:id="butonRenuntare" mnemonicParsing="false"
onAction="#handleButonRenuntareAction" text="Renuntare" />
          </children>
          <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
          </padding>
        </HBox>
        <HBox id="rezultat" prefHeight="100.0" prefWidth="200.0"
GridPane.columnIndex="0" GridPane.rowIndex="3">
          <children>
            <Label fx:id="etichetaAfisare" textFill="RED" />
          </children>
        </HBox>
      </children>
    </GridPane>
  </children>
  <stylesheets>
    <URL value="@authentication.css" />
  </stylesheets>
</AnchorPane>
```

Ierarhia nodurilor din documentul FXML descris are următoarea structură

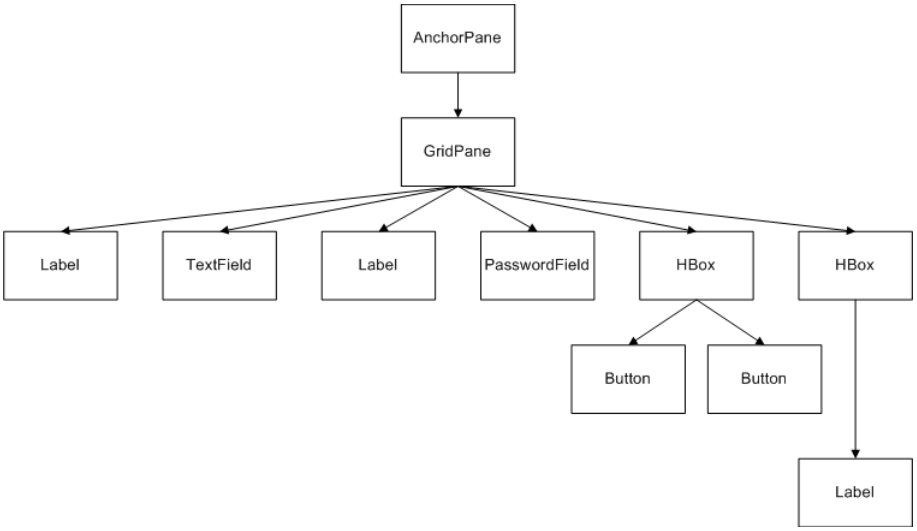


Figura 12 – Ierarhia nodurilor pentru un document FXML corespunzător unei interfețe cu utilizatorul pentru procesul de autentificare

Clasa `BookStore` (controller) va fi responsabilă cu gestiunea evenimentelor asociate nodurilor, trebuind să definească cel puțin metodele `handleButonAcceptareAction` şi `handleButonRenuntareAction`. Acestea vor trebui adnotate cu şirul de caractere `@FXML`, ca şi obiectele corespunzătoare nodurilor descrise în documentul respectiv.

Încărcarea documentului într-o scenă se va face astfel:

```
try {
    applicationScene =
    new Scene(
    (Parent)FXMLLoader.load(getClass().getResource("authentication.fxml"))
    );
} catch (Exception exception) {
    System.out.println ("exception : "+exception.getMessage());
}
```

7. Scene Builder: dezvoltarea de interfețe grafice JavaFX în mod vizual

Documentele FXML pot fi dezvoltate în mod vizual folosind utilitarul SceneBuilder 1.0, disponibil pentru platformele Windows (32 şi 64 de biți), respectiv MacOS. Acesta permite vizualizarea ierarhiei de noduri precum şi stabilirea de diferite aspecte pentru fiecare nod în parte, grupate în proprietăți, moduri de dispunere şi cod.

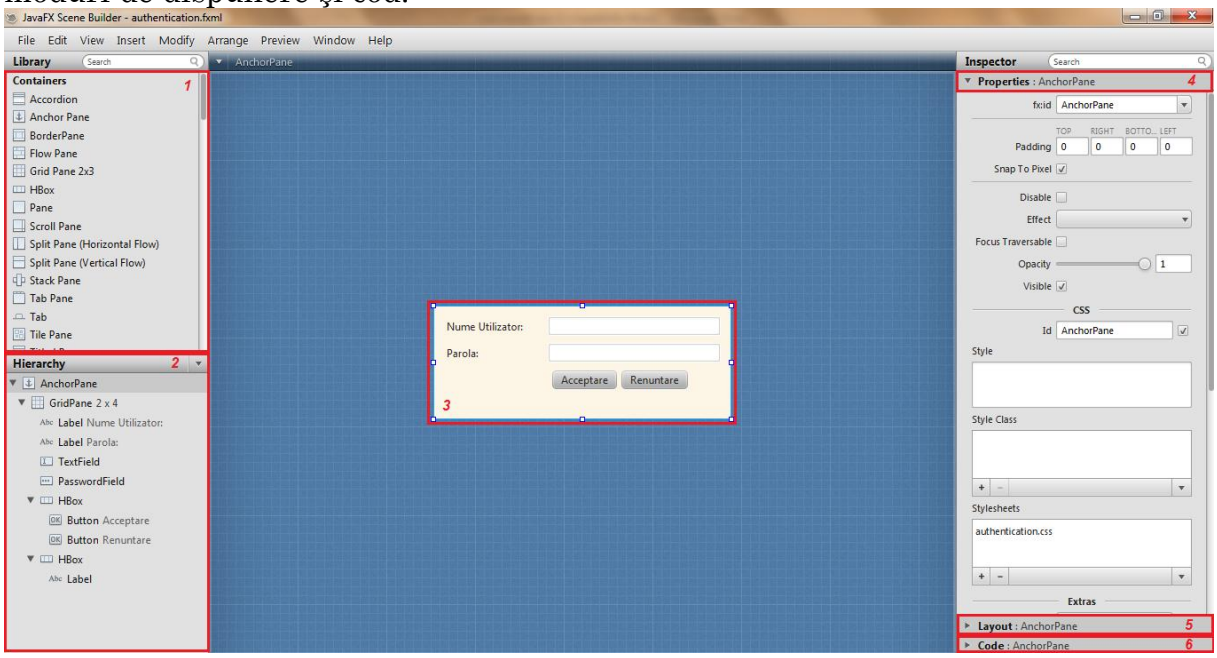


Figura 13 – Interfața SceneBuilder pentru o interfață cu utilizatorul corespunzătoare unui proces de autentificare

În secțiunea 1 sunt disponibile un set de obiecte (noduri) cu care se poate construi interfața cu utilizatorul. Ele pot indica modul de dispunere al obiectelor sau pot constitui controale propriu-zise.

În secțiunea 2 este prezentată ierarhia de noduri. Un obiect care conține şi alte obiecte la rândul său va avea asociată o săgeată având vârful orientat în jos, acesta putând fi utilizat pentru expandarea nodurilor corespunzătoare.

În secțiunea 3 se poate previzualiza interfața cu utilizatorul. De asemenea, aici aceasta se construieşte propriu-zis prin operații de tip drag and drop. Dacă nodul care se adaugă interfeței se află deasupra unui alt obiect[77], acesta va fi evidențiat în mod deosebit, astfel încât să se poată determina cu exactitate ierarhia nodurilor din interfața cu utilizatorul.

În secțiunea 4 se pot stabili proprietățile obiectului selectat în mod curent (identificator, efecte, transformări, asocierea unor foi de stil).

[77] În situația în care un nod este adăugat peste un alt obiect, acesta va fi inclus în nodul respectiv, stabilindu-se o relație de tip părinte-fiu.

În secțiunea 5 sunt tratate aspecte legate de modul de dispunere a nodului (dimensiuni, aliniere).

În secțiunea 6 sunt indicate metodele care vor fi executate din clasa controller atunci când o acțiune de tipul respectiv se produce asupra interfeței respective[78].

Utilitarul SceneBuilder va construi documentul FXML corespunzător, care poate fi integrat ulterior în proiectul JavaFX.

Activitate de Laborator

Pentru rezolvarea acestui laborator, trebuie să aveți instalate:

Java 7, update > 6 (JavaFX 2.2.3)

NetBeans 7.2 / Eclipse 4.2.1[79] (Juno)

Scene Builder 1.0

Se doreşte implementarea unei interfețe grafice cu utilizatorul pentru gestiunea informațiilor dintr-o bază de date, aceasta urmând a fi utilizată pentru un sistem ERP destinat unei librării care comercializează doar cărți.

Aplicația va avea inițial două ferestre: un formular pentru autentificarea utilizatorilor în sistem, respectiv o interfață grafică în care sunt implementate operațiile pentru o tabelă dintr-o bază de date (adăugare, modificare, ştergere, căutare).

Structura formularului pentru autentificarea utilizatorilor în sistem este descrisă într-un document FXML generat cu SceneBuilder.

Figura 14 – Formularul de Autentificare

[0p] 0. Să se ruleze scriptul `Laborator04.sql` în MySQL Workbench. În clasa `general.Constants`, să se modifice corespunzător proprietățile `DATABASE_USER`, respectiv `DATABASE_PASSWORD`.

[2p] 1. Să se deschidă fişierul authentication.fxml cu SceneBuilder 1.0 (Windows) sau cu orice alt editor de text (Linux) – inclusiv în cadrul NetBeans / Eclipse.

a. Pentru nodul rădăcină de tip `AnchorPane`, să se indice drept controller clasa `BookStore`.
b. Pentru butoanele `Acceptare` / `Renunţare`, să se specifice metodele care vor fi executate în cazul producerii unui eveniment de tipul `ActionEvent.ACTION`.

[78] Categoriile de evenimente incluse sunt drag and drop, evenimente legate de tastaura şi mouse, precum şi evenimente specifice dispozitivelor cu ecran tactil (derulare, rotație, atingere, modificarea rezoluției de vizualizare).

[79] Pentru proiectul Eclipse, se recomandă ca biblioteca `jfxrt.jar` să fie preluată de la `%SystemRoot%\Program Files\Java\jdk1.7.0_0x\jre\lib` [x=6..9], respectiv de la `$JAVA_HOME\jdk1.7.0_0x \jre\lib`.

c. În clasa `BookStore`, sa se implementeze metodele `handleButonAcceptareAction` / `handleButonRenuntareAction` ce permit sau nu accesul utilizatorului în aplicație. Vor avea acces în sistem doar utilizatorii al căror rol este *administrator*.

   În cazul în care combinația (utilizator, parolă) este incorectă sau utilizatorul nu are acces în sistem, se va afişa un mesaj corespunzător în `etichetaAfisare`. Obiectele şi metodele definite în documentul FXML pot fi accesate din clasa controller daca sunt adnotate cu şirul de caractere `@FXML`.

[2p] 2. În clasa DataBaseManagementGUI, să se creeze un obiect de tip `GridPane` care va avea pe două coloane obiectele de tip `ArrayList<Label>` (`attributeLabels`), respectiv `ArrayList<Control>` (`attributeControls`).

Figura 15 – Detalii cu privire la o înregistrare în tabela cărți

Înainte de a adăuga obiectul propriu-zis, trebuie să îi specificați poziția în cadrul obiectului de tip `GridPane`, prin intermediul metodei `GridPane.setConstraints(object,col,row);`

În interfața grafică se folosesc trei butoane (adăugare, editare, ştergere) pentru a se realiza operațiile respective. În momentul când se produce o acțiune corespunzătoare unuia dintre aceste butoane, se va apela metoda `handle` specifică metodei de tratare a evenimentului, asociată printr-un mecanism de oportunitate (*eng.* convenience method) – metoda `setOnMouseClicked`.

[2p] 3. Să se implementeze metoda `handle` corespunzătoare butonului care actualizează anumite valori în tabela.

[1p] 4. Să se implementeze metoda `handle` corespunzătoare butonului care crează o înregistrare nouă, fără a completa alte atribute cu excepția cheii primare[80].

[2p] 5. Să se creeze un submeniu *Despre...* al meniului *Ajutor* care în momentul când este accesat deschide o fereastră în care sunt afişate numele aplicației, versiunea şi anul în care a fost realizată. Totodată, aceasta va conține un buton care va închide fereastra atunci când este apăsat.

[2p] 6. Să se creeze un buton *Căutare* la apăsarea căruia vor fi afişate în tabel toate înregistrările care corespund unor anumite criterii (au anumite valori corespunzătoare unor anumite atribute, specificate de utilizator în câmpurile text şi listele derulante).

[1p] 7. Aplicația generează excepții într-o anumită situație. Identificați contextul şi corectați problema.

[80] Această regulă se aplică doar în cazul cheilor primare autoincrementale, deci nu şi pentru tabela utilizatori.

Bibliografie

[1] JavaFX Overview, http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm
[2] JavaFX Architecture and Framework,
http://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm
[3] Using JavaFX UI Controls,
http://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm
[4] Using Text and Text Effects in JavaFX,
http://docs.oracle.com/javafx/2/text/jfxpub-text.htm
[5] Handling JavaFX Events, http://docs.oracle.com/javafx/2/events/jfxpub-events.htm
[6] Mastering FXML,
http://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm