

Aplicații Integrate pentru Întreprinderi

Laborator 4

28.10.2013

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 și înțelegerea arhitecturii pe baza căreia a fost construită aceasta.

1. Ce este JavaFX ?
2. Arhitectura JavaFX
3. Structura unei aplicații JavaFX
4. Construirea de interfețe grafice cu ajutorul controalelor JavaFX
5. Tratarea evenimentelor asociate controalelor JavaFX
6. Utilizarea de elemente grafice avansate și conținut multimedia
7. Utilizarea FXML în JavaFX
8. Scene Builder: dezvoltarea de interfețe grafice JavaFX în mod vizual
9. Modele de date asociate controalelor JavaFX
10. Gestiunea concurenței în JavaFX
11. Câteva recomandări cu privire la implementarea aplicațiilor JavaFX

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 și motoare grafice care facilitează crearea unor aplicații pentru organizații bazate pe date. Fiind implementat peste limbajul de programare Java, acesta va fi ușor de utilizat, optimizând costurile legate de investiții și complexitatea sistemelor, întrucât atât componenta server (nivelul de logică a aplicației și de acces la date) cât și componenta client (nivelul de prezentare) vor fi dezvoltate utilizând același limbaj de programare.

Câteva dintre caracteristicile JavaFX 2.2 sunt următoarele [1]:

- **integrare cu Java.** JavaFX¹ reprezintă o interfață de programare integrată în distribuția limbajului de programare Java, instalarea lor făcându-se împreună; prin urmare, din JavaFX pot fi accesate toate API-urile Java cu funcționalitățile lor, acesta reprezentând o alternativă la celelalte limbaje de programare care folosesc Java Virtual Machine, precum JRuby sau Scala;
- **limbajul de adnotare FXML și utilitarul SceneBuilder.** FXML este un limbaj declarativ de adnotare bazat pe XML prin intermediul căruia pot fi dezvoltate interfețe grafice cu utilizatorul în mod interactiv, fără a fi necesar ca aplicația să fie recompilată de fiecare dată când sunt modificate elemente din cadrul acesteia. SceneBuilder permite construirea interfeței în mod vizual, generând automat și documentul FXML asociat, acesta putând fi integrat apoi în orice mediu de dezvoltare, pentru implementarea nivelului de logică a aplicației;

¹ În prezent, SDK-ul Java 7 a ajuns la versiunea update 45, iar cel de JavaFX la versiunea 2.2.45 (lansată în octombrie 2013). De asemenea, se pregătește o distribuție JavaFX care va fi integrată cu Java 8.

- **componente WebView.** Un modul ce folosește tehnologia WebKitHTML permite integrarea de pagini Internet în cadrul unei interfețe grafice. De asemenea, codul JavaScript ce rulează în WebView poate accesa orice API-uri Java, la fel cum API-urile Java pot executa cod JavaScript care rulează în WebView;
- **interoperabilitate cu tehnologia Swing.** Aplicațiile Swing existente pot fi extinse cu funcționalități JavaFX, ca redarea de conținuturi grafice sau a unor pagini Internet integrate;
- **controale integrate și foi de stil.** JavaFX pune la dispoziția programatorilor majoritatea controalelor necesare pentru dezvoltarea unei interfețe cu utilizatorul complete, ele putând fi modificate folosind tehnologii web standard ca foile de stil (*eng.* CSS – cascading style sheets);
- **API-ul Canvas** permite desenarea pe o suprafață dintr-o scenă JavaFX conținând un element grafic (de tip nod);
- **suport pentru operații multi-touch.** În cazul sistemelor încorporate, JavaFX folosește capacitățile puse la dispoziție de platformele în cauză pentru a procesa mai multe evenimente de tip atingere concurente;
- **grafică accelerată hardware la nivel de bandă de asamblare.** JavaFX dispune de un motor pentru redarea conținutului grafic denumit Prism care oferă performanțe deosebite atunci când este utilizat împreună cu anumite plăci grafice sau unități de procesare grafică (*eng.* GPU – Graphic Processing Unit)²;
- **motor multimedia performant,** pentru redarea de conținut multimedia folosind tehnologia GStreamer, caracterizată prin latență scăzută și stabilitate;
- **model de dezvoltare al aplicațiilor autonome.** Există pachete de aplicații independente care dispun de toate resursele necesare precum și de copii ale mediilor de rulare Java și JavaFX distribuite ca pachete native ce pot fi instalate oferind utilizatorilor aceeași experiență ca și celelalte aplicații native sistemului de operare respectiv.

JavaFX este folosit pentru aplicații care folosesc rețeaua de calculatoare, capabile să ruleze pe mai multe platforme simultan, în care informațiile pot fi vizualizate folosind o interfață cu utilizatorul performantă care folosește capacități audio și video (inclusiv tehnici de grafică avansată și animații).

Aplicații JavaFX pot fi rulate fie direct pe o anumită mașină, unde se află „instalată”³ (accesând fișierul .jar care conține programul în sine), fie poate fi accesată prin intermediul unui navigator (*eng.* browser), fiind integrată în cadrul unei pagini Internet⁴. Totodată, se poate ca sistemul informatic să fie descărcat de la o anumită locație, rulând apoi pe mașina respectivă (modul WebStart⁵). Aplicația JavaFX poate fi executată și ca program autonom, folosind copii private ale mediilor de rulare Java și JavaFX.

² În cazul în care un sistem nu dispune de una dintre dispozitivele recomandate pentru a rula împreună cu JavaFX, Prism va folosi în mod implicit stiva de aplicații Java2D.

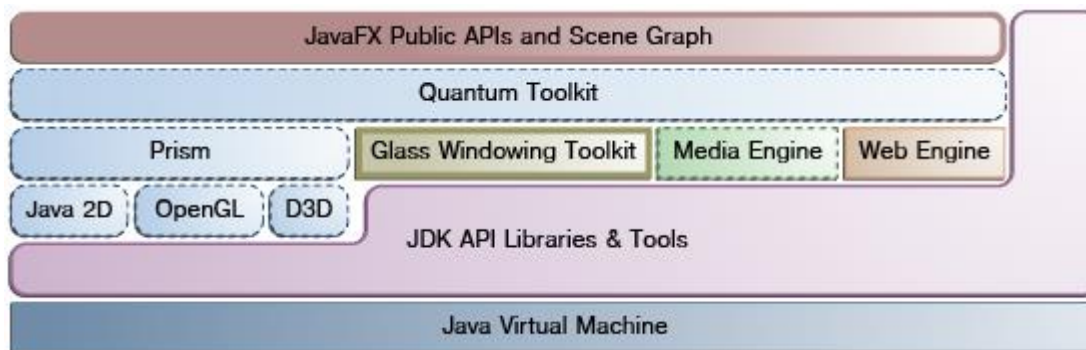
³ O astfel de opțiune este preferată în cazul în care aplicația nu are nevoie de acces Internet sau mașina pe care rulează nu dispune de o astfel de conexiune.

⁴ Interacțiunea cu elementele paginii Internet se poate face prin intermediul limbajului JavaScript.

⁵ În această situație, aplicația JavaFX va fi găzduită pe un server web.

2. Arhitectura JavaFX

Platforma JavaFX are de o arhitectură bazată pe o stivă de componente⁶, constituind motorul ce rulează codul propriu-zis (Quantum Toolkit). Acesta cuprinde motorul grafic de performanță înaltă (Prism), sistemul de ferestre (Glass), un motor pentru redarea conținutului multimedia și un motor pentru integrarea conținutului Internet.



Arhitectura JavaFX

La nivelul cel mai înalt al arhitecturii JavaFX se află **graful de scene** (*eng.* Scene Graph), structură ierarhică de noduri⁷ ce conține elementele vizuale ale interfeței grafice cu utilizatorul, care poate trata evenimente de intrare și care poate fi redată. Un element este identificat în mod unic, fiind caracterizat printr-o clasă de stil și un volum care îl delimitează. Fiecare nod are un părinte⁸, putând avea nici unul sau mai mulți copii. De asemenea, pentru un astfel de element pot fi definite efecte (estompări sau umbre), opacitate, transformări, mecanisme de tratare a diferitelor evenimente (care vizează interacțiunea cu utilizatorul) precum și starea aplicației. Spre diferență de Swing sau AWT (Abstract Window Toolkit), JavaFX conține și primitive pentru elemente grafice⁹, mecanisme de dispunere a conținutului, controale, imagini și obiecte multimedia.

API-ul `javafx.scene` permite construirea următoarelor conținuturi:

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

Platforma JavaFX pune la dispoziție și **un set de API-uri** care permit implementarea unor aplicații Internet¹⁰ într-un mod flexibil, exploatând capacitățile limbajului de programare Java pentru a dezvolta intuitiv funcționalități referitoare la conținut multimedia. Se permite de asemenea integrarea cu alte limbaje dinamice (Groovy, JavaScript) din care pot fi accesate facilitățile JavaFX. Mecanismul de tipare permite evaluare leneșă performantă, inclusiv pentru expresii și secvențe de expresii prin intermediul sintaxei.

⁶ Componentele JavaFX funcționează transparent, nefiind accesibile propriu-zis programatorilor.

⁷ Prin termenul de nod se desemnează un element din graful de scene.

⁸ Excepție de la această regulă o face nodul rădăcină.

⁹ Astfel de primitive grafice pot fi texte sau figuri geometice cu care se pot crea animații, folosind metodele puse la dispoziție de API-urile `javafx.animation`.

¹⁰ Platforma JavaFX se bazează pe numeroase standarde Internet, cum ar fi CSS pentru stiluri sau ARIA pentru specificări referitoare la accesibilitate.

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 și 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)¹¹ folosind DirectX 9 (Windows XP / Vista), DirectX 11 (Windows 7), OpenGL (Mac, Linux, sisteme încorporate), Java 2D¹² (dacă hardware-ul nu dispune de accelerare grafică);
- **Quantum Toolkit** reprezintă un element de interfață între Prism și sistemul de ferestre Glass pe care le face disponibile nivelului JavaFX, ocupându-se și de prioritatea firelor de execuție ce tratează redarea față de tratarea evenimentelor de intrare/ieșire.

În arhitectura JavaFX, la nivelul cel mai scăzut se află **Glass – un sistem de ferestre**, în fapt o interfață între API-ul JavaFX și sistemul de operare care gestionează ferestrele, mecanisme de cronometrare, suprafețe, evenimentele¹³ de intrare / ieșire.

Platforma JavaFX rulează pe mai multe fire de execuție la un moment dat (minim 2):

- firul de execuție JavaFX utilizat pentru a gestiona scenele care fac parte din ferestrele care sunt afișate¹⁴;
- firul de execuție Prism folosit pentru redare, separat de gestiunea evenimentelor de intrare / ieșire – permițându-se ca un cadru să fie afișat în timp ce alt cadru este procesat în fundal¹⁵; de asemenea, poate asocia alte fire de execuție folosite pentru rasterizare, eliminând încărcarea diun procesul de randare propriu-zis;
- un fir de execuție pentru conținut multimedia, ce rulează în fundal, utilizat pentru sincronizarea celor mai recente cadre din graful de scene utilizând firul de execuție JavaFX.

¹¹ Prism suportă atât redare hardware (când sistemul pe care rulează aplicația dispune de acceleratoare grafice) cât și redare software (când echiparea mașinii în cauză este insuficientă). Prism implementează și redarea de obiecte 3D.

¹² 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. Totuși, în acest caz, performanțele sunt mai reduse decât în cazul în care se folosește randarea la nivel hardware.

¹³ Spre diferență de AWT care folosește o coadă de evenimente de intrare/ieșire 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 același fir de execuție ca și aplicația JavaFX (în AWT exista un fir de execuție specific bibliotecii și un fir de execuție specific Java, ceea ce crea o serie de probleme).

¹⁴ Un graf de scene poate fi creat într-un fir de execuție separat, însă atunci când nodul 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. De asemenea, firul de execuție al aplicației JavaFX este direrit de firul de execuție asociat evenimentelor Swing sau AWT, astfel încât atunci când se integrează astfel de cod în aplicațiile JavaFX, gestiunea obiectelor create în fire de execuție diferite trebuie să se realizeze cu mare atenție.

¹⁵ Procesarea concurentă reprezintă un avantaj în special în contextul sistemelor care dispun de mai multe procesoare, astfel încât fiecare fir de execuție poate fi asociat unui procesor diferit, îmbunătățind viteza de rulare a aplicației și experiența utilizatorului.

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 declanșează la maxim 60 de cadre pe minut sau oricând sunt redade diferite animații¹⁶, actualizând starea elementelor din graful de scene, evenimentele fiind tratate asincron. Se permite astfel ca evenimentele să fie tratate în funcție de puls¹⁷. 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 auditiv¹⁸ cât și vizual¹⁹. Sunt implementate trei componente: obiectul `Media` care reprezintă fișierul, obiectul `MediaPlayer` care redă conținutul acestuia și obiectul `MediaView` reprezentând nodul ce afișează utilizatorilor obiectul multimedia propriu-zis. Motorul multimedia a fost proiectat pentru a oferi stabilitate, performanță și comportament consistent pentru toate platformele.

JavaFX include și 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 și SVG, oferind o funcționalitate asemănătoare cu cele mai multe produse de același tip (redare conținut HTML local și aflat într-o locație la distanță, implementare istoric cu navigare înainte și înapoi în cadrul acestuia, reîncărcarea conținutului, aplicarea unor efecte, editarea conținutului HTML, execuția de comenzi JavaScript și tratarea evenimentelor). La nivelul componentei care integrează navigatorul sunt definite două clase: `WebEngine` care oferă funcționalități de bază caracteristice unui navigator și `WebView`²⁰ care încapsulează un obiect `WebEngine`, încorporând conținut HTML în scena aplicației, oferind atribute și metode pentru aplicarea de efecte și transformări. De asemenea, apelurile Java pot fi controlate prin intermediul JavaScript și invers, astfel încât utilizatorii pot beneficia de avantajele ambelor medii de programare.

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 aplicate²¹ asupra oricărui obiect de tip `Node` din graful de scene. Fișierele conținând foi de stil pot fi parsate de orice utilitar, chiar dacă acesta nu este integrat cu JavaFX. Integrarea JavaFX / CSS poate fi utilizată și pentru pagini HTML. Toate proprietățile JavaFX sunt prefixate cu șirul `-fx-` chiar și pentru cele compatibile CSS HTML, datorită faptului că semantica poate fi diferită pentru valorile corespunzătoare unor atribute.

¹⁶ Un puls poate fi generat chiar și atunci când nu sunt redade animații, însă există modificări în graful de scene.

¹⁷ Actualizarea disponibilității 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 (disponere conținut / stil) pentru a deveni vizibile înaintea unui puls.

¹⁸ Sunt implementate formatele `.mp3`, `.aiff` și `.wav`.

¹⁹ Este suportat formatul `.flv` și `.mp4`.

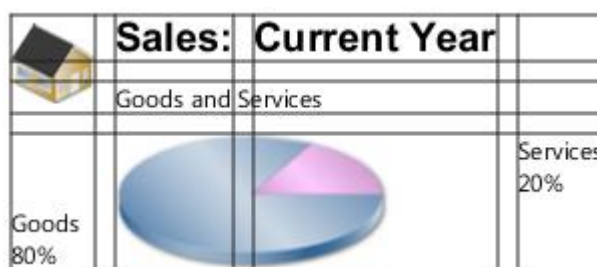
²⁰ `WebView` reprezintă o extensie a clasei `Node`.

²¹ Aplicarea se face în mod asincron și se poate realiza inclusiv la runtime, permițând modificarea conținutului în mod dinamic. Este implementată specificația W3C CSS versiunea 2.1.

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 pentru dispunerea elementelor definește mai multe clase de tip container predefinite în pachetul `javafx.scene.layout`:

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



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.

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 coordonate Oxyz folosind metodele puse la dispoziție de 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` – rotește o axă astfel încât axele x sau y să nu mai fie perpendiculare coordonatele nodului fiind modificate conform specificațiilor;
- `rotate` – rotește un nod în jurul unui punct, denumit pivot al scenei;
- `affine`²² – 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.

²² Această metodă ar trebui să fie folosită împreună cu clasele de transformare `Translate`, `Scale`, `Rotate`, `Shear` mai degrabă decât să fie apelate direct.

Îmbogățirea aplicațiilor JavaFX (în special a celor de timp real) se face prin efecte vizuale care operează la nivel de pixel al imaginii²³, cum ar fi Bloom (face ca zona cea mai luminoasă a unei imagini să pară strălucitoare), Blur (adaugă un efect de neclaritate asupra obiectului, putând fi de tipul BoxBlur, MotionBlur sau GaussianBlur), DropShadow / InnerShadow (adaugă o umbră în spatele / în interiorul conținutului respectiv), Reflection (creează o versiune reflectată a conținutului respectiv, dispunând-o sub acesta), Lighting (simulează o sursă de lumină care operează asupra unui conținut, dând un aspect mai realist, 3D, asupra unui obiect plat) și Perspective (mapează un dreptunghi peste un alt dreptunghi, menținând netezimea liniilor dar nu neapărat și paralelismul acestora).

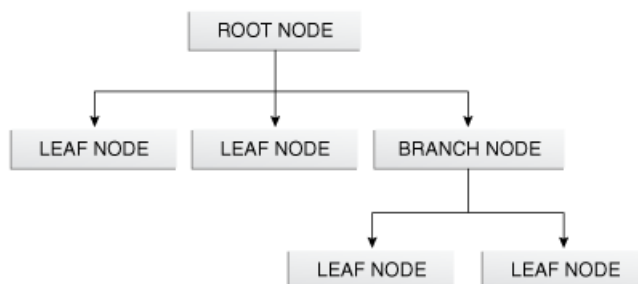
3. Structura unei aplicații JavaFX



Containere într-o aplicație 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`²⁴ și `Scene`²⁵.

Fiecare element din cadrul unei scene este exprimat sub forma unui nod dintr-un graf (arbore) în care elementul rădăcină este reprezentat de containerul din care acestea fac parte.



Tipuri de noduri în graful (arborele) de scene

Pachetul `javafx.scene` definește clase aferente celor trei tipuri de noduri care pot apărea în graful (arborele) de scene:

- `Scene`: containerul de bază pentru întregul conținut (nod rădăcină);
- `Parent`: clasă abstractă pentru nodurile ce pot avea copii (nod ramură); implementări ale acestei clase sunt `Control`, `Group`, `Region` și `WebView`;
- `Node`: clasă abstractă pentru toate nodurile, folosit în special pentru nodurile care nu pot avea copii (nod frunză); clasele ce pot fi folosite în acest scop sunt definite mai cu seamă în pachetele `javafx.scene.shape` și `javafx.scene.text`.

²³ Toate nodurile din graful de scene sunt randate ca imagini, iar efectele sunt aplicate asupra acestor imagini.

²⁴ Clasa `Stage` reprezintă container-ul de nivel înalt.

²⁵ Clasa `Scene` reprezintă container-ul pentru toate elementele. Acesta are asociat graful de noduri menținând un model intern al obiectelor grafice din cadrul aplicației. La fiecare moment de timp, acesta știe ce obiecte să afișeze, care zone ale ecranului trebuie actualizate și cum să realizeze acest proces într-un mod eficient. Metodele de desenare nu sunt apelate manual de utilizator, ci automat de sistemul de randare, conform informațiilor reținute în graful de noduri.

Aceste clase de bază definesc funcționalități importante care vor fi moștenite de subclasele lor, inclusiv ordinea de desenare, vizibilitatea, compunerea transformărilor sau suportul pentru stiluri CSS.

În exemplul de mai jos, aplicația JavaFX poartă numele `BookStore` și extinde clasa `javafx.application.Application`. Metoda `start` primește un argument de tip `javafx.stage.Stage` ce reprezintă container-ul de nivel înalt. În cadrul acesteia se definește o scenă având ca element rădăcină un element `VBox` (care identifică un mod de dispunere pe verticală a nodurilor componente) și 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 rădăcină.

```
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 screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        sceneWidth  = Constants.SCENE_WIDTH_SCALE*screenSize.width;
        sceneHeight = Constants.SCENE_HEIGHT_SCALE*screenSize.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 fișierul `.jar` corespunzător aplicației JavaFX este creat folosind utilitarul JavaFX Packager, care include și un mecanism pentru lansarea aplicației în cadrul acesteia (JavaFX launcher). Totuși, metoda `main()` va trebui implementată atunci când nu se utilizează acest utilitar (IDE-uri care nu au integrat JavaFX²⁶).

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 CSS²⁷ fie utilizând clasa `Skin`. Totodată, ele pot fi integrate cu ușurință cu alte aplicații Java²⁸. Clasele care implementează controale din JavaFX se găsesc în pachetul `javafx.scene.control`. Pe lângă controalele clasice (existente în AWT și Swing) au mai fost dezvoltate și alte componente, cum ar fi `Accordion`, `ColorPicker`, `Chart`²⁹, `Pagination` sau `TitledPane`.

²⁶ În această situație, din metoda `main` va trebui apelată metoda `Application.launch()`.

²⁷ Utilizarea foilor de stil (CSS) în aplicațiile JavaFX este similară ca și în HTML întrucât folosesc aceeași specificație CSS. Pentru fiecare obiect instanță a unui control se poate apela metoda `setStyle` primind ca parametru un șir 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 fișier `.css` care specifică formatul în care sunt vizualizate diferite elemente este asociat unei scene prin metoda `scene.getStyleSheets().add("...css")`.

²⁸ 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 Swing respectivă. Chiar și în acest caz, elementele JavaFX sunt randate cu Prism.

²⁹ În `javafx.scene.chart` sunt definite mai multe tipuri de grafice (`area`, `bubble`, `line`, `pie`, `scatter`) acestea putând conține mai multe seturi de date.

Fiind derivate din clasa `Node`, controalele pentru interfața cu utilizatorul pot fi integrate în diferite animații, efecte³⁰, 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 și metode (suplimentare față de cele din clasa `Control`) spre a oferi suport intuitiv pentru interacțiunea cu utilizatorul.



Tipuri de controale definite în JavaFX

Un obiect de tip `Label` poate primi ca argument textul asociat³¹ precum și imaginea (obiect de tip `ImageView`) care va fi afișată împreună cu acesta. Parametrii pot fi specificați și prin intermediul unor metode:

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

Când se folosește atât text cât și imagine pentru o etichetă, distanțarea dintre ele se poate face folosind metoda `setGraphicTextGap`. Alinierea textului în cadrul etichetei utilizează metoda `setTextAlignment`, iar definirea unei poziții a elementului grafic față de text se face prin metoda `setContentDisplay` care poate primi ca argument una din constantele clasei `ContentDisplay` și anume: `LEFT`, `RIGHT`, `CENTER`, `TOP`, `BOTTOM`. În situația în care textul trebuie împărțit între mai multe rânduri, metoda `setWrapText` va primi ca argument valoarea `true`.

³⁰ Efectele vizuale disponibile în pachetul `javafx.scene.effect` sunt umbrirea, iluminarea sau estomparea imaginii.

³¹ Culoarea cu care va fi afișat textul respectiv se stabilește prin metoda `setTextFill()`, iar indicarea unui font se face prin metoda `setFont()`.

Totodată, în cazul în care textul asociat nu poate fi vizualizat datorită dimensiunilor etichetei respective, se folosește metoda `setTextOverrun` care primește ca argument unul dintre tipurile clasei `OverrunStyle` ce definește modul în care va fi procesat textul care nu a putut fi vizualizat corespunzător.

Efectele care pot fi aplicate asupra obiectelor de tip `Labeled`, ca de altfel asupra tuturor controalelor din `JavaFX` sunt rotirea (`setRotate`), translația (`setTranslate`) și mărirea/micșorarea (`setScale`), numele metodelor putând fi sufixate de axa ce servește drept referință a operației respective.

Și obiectele de tip `Button` suportă aceleași metode, întrucât sunt derivate din clasa `Labeled`. Funcția unui buton este de a produce o acțiune în momentul în care este apăsat. Execuția acestei operații se face prin asocierea unui obiect `EventHandler`³², transmis ca parametru al metodei `setOnAction`. Cel mai frecvent, tipul de eveniment asociat este `ActionEvent`:

```
button.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        // ...  
    }  
});
```

De asemenea, asupra unui astfel de obiect pot fi aplicate și efecte (pachetul `javafx.scene.effect`) prin metoda `setEffect` asociate de regulă evenimentelor legate de mouse, cum ar fi `MouseEvent.MOUSE_ENTERED` și `MouseEvent.MOUSE_EXITED`.

Stilizarea unui buton se face apelând metoda `setStyleClass` ce primește drept argument un fișier de tip `.css` sau prin metoda `setStyle` în care elementele specifice de stil sunt precizate manual.

Aceleași metode pot fi folosite pentru obiecte de tip `RadioButton`, însă acestea trebuie grupate în cadrul unui obiect de tip `ToggleGroup`, adaugarea într-un astfel de grup făcându-se prin metoda `setToggleGroup` care primește ca argument elementul din care va face parte.

Selecția unui anumit buton din grup trebuie realizată atât prin metoda `setSelected(true)` cât și prin metoda `requestFocus`. Verificarea faptului că butonul a fost selectat se face prin metoda `isSelected`.

Identificarea butonului selectat în cadrul interacțiunii cu utilizatorul implică asocierea unui obiect ascultător de tip `ChangeListener`:

```
final ToggleGroup tGroup = new ToggleGroup();  
group.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {  
    public void changed(ObservableValue<? extends Toggle> oValue,  
                       Toggle oldToggle, Toggle newToggle) {  
        if (group.getSelectedToggle() != null) {  
            Toggle toggle = group.getSelectedToggle();  
            // ...  
        }  
    }  
});
```

Un tip particular de butoane asemănător cu `RadioButton` este `ToggleButton`, fiind de asemenea asociat unui grup³³.

³² Un obiect de tip `EventHandler` implementează metoda `handle` care realizează procesările asociate acțiunii de apăsare a unui buton.

³³ Distincția dintre un obiect `RadioButton` și un obiect `ToggleButton` este că în cazul `RadioButton` un buton din grup este selectat pe când în cazul `ToggleButton` există și posibilitatea ca nici un buton din grup să nu fie selectat la un moment dat de timp.



Modele de obiecte de tip Label, Button și RadioButton

Obiectele de tip `CheckBox` nu pot fi grupate în obiecte `ToggleGroup` întrucât mai multe obiecte de acest tip pot fi selectate simultan.

Un obiect `CheckBox` poate fi definit (situație în care acesta poate fi selectat sau deselectat) sau nedefinit, caz în care obiectul nu poate fi modificat ca urmare a interacțiunii cu utilizatorul. Definirea obiectului presupune specificarea proprietăților sale prin metodele `setSelected` respectiv `setIndeterminate`.

Stările unui obiect de tip `CheckBox`

	INDETERMINATE=false ³⁴	INDETERMINATE=true ³⁵
SELECTED=false	<input type="checkbox"/> I agree	<input type="checkbox"/> I agree
SELECTED=true	<input checked="" type="checkbox"/> I agree	

Dacă utilizatorul dorește ca obiectul `CheckBox` să cicleze între trei valori: selectat, neselectat și nedefinit, se precizează proprietatea `allowIndeterminate`.

Prin intermediul obiectelor de tip `ChoiceBox` 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`. Totodată, nodul poate fi însoțit de o explicație suplimentară, folosind metoda `setTooltip` (care primește drept parametru un obiect de tipul `Tooltip`).

Identificarea indexului pe care îl deține elementul selectat în cadrul interacțiunii cu utilizatorul se face tot prin intermediul unui obiect ascultător de tipul `ChangeListener`:

```
choiceBox.getSelectionModel().selectedIndexProperty().addListener(
    new ChangeListener<Number>() {
        public void changed(ObservableValue oValue,
                           Number oldValue, Number newValue) {
            int selectedIndex = newValue.intValue();
        }
    }
);
```

Metoda `getSelectionModel` întoarce elementul selectat din cadrul obiectului `choiceBox`, în timp ce metoda `selectedIndexProperty` returnează proprietatea `SELECTED_INDEX` a aceluiași obiect.

³⁴ În cazul în care proprietatea `INDETERMINATE` are valoarea `false`, starea unui obiect `CheckBox` poate fi selectat sau deselectat.

³⁵ În cazul în care proprietatea `INDETERMINATE` are valoarea `true`, starea unui obiect `CheckBox` poate fi selectat, deselectat sau nedefinit.

Clasele `TextField` și `PasswordField`, derivate din clasa `TextInput` oferă posibilitatea de a primi un text de la utilizator³⁶ pe care îl poate afișa. 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 conținutului³⁷ este indicat prin metoda `setPromptText`. Ștergerea textului conținut de aceste obiecte se face apelând metoda `clear`. 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, nementinându-l în obiectul curent;
- `paste()` – transferă conținutul zonei de memorie în obiectul curent, înlocuind textul selectat.

Obiectele `ScrollBar` permit crearea de panouri și zone de vizualizare care pot fi derulate. Elementele sale sunt cursorul, bara culisantă precum și butoanele stâng și drept. Câteva dintre metodele caracteristice acestui tip de element sunt `setMin` și `setMax` pentru a exprima limitele între care cursorul se mișcă, `setValue` pentru a specifica poziția curentă a cursorului. Implicit, cursorul este centrat pe orizontală. Metoda `setOrientation` este folosită pentru a distinge între tipurile orizontal și 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 (respectiv sus și jos). Atributul `BLOCK_INCREMENT` indică deplasarea implicită în momentul în care utilizatorul apasă bara culisantă în loc de butoanele de la extremitățile acesteia.

Obiectele de tip `ScrollPane` oferă vizualizări ale controalelor în zone care pot fi derulate folosind elemente culisante (`ScrollBar`). Conținutul (nodul³⁸) unui astfel de obiect este specificat prin intermediul metodei `setContent`. Se pot specifica politici de afișare a elementelor derulante, atât pentru verticală cât și pentru orizontală prin metodele `setHbarPolicy` / `setVbarPolicy`. Parametrii pe care îi pot primi 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`)³⁹. Proprietățile `HVALUE` și `VVALUE` ale obiectului `ScrollPane` ajută la identificarea poziției pe care o are cursorul de derulare pe orizontală, respectiv pe verticală.

³⁶ Diferența dintre clasele `TextField` și `PasswordField` este că în cazul `TextField` conținutul e afișat în timp ce pentru `PasswordField` conținutul nu e afișat. De asemenea, o practică frecventă pentru obiecte de tip `PasswordField` este ștergerea conținutului său după ce a fost realizată autentificarea.

³⁷ Diferența dintre conținutul introdus de utilizator într-un obiect `TextField` / `PasswordField` și textul explicativ este că doar valoarea introdusă poate fi obținută prin metoda `getText`. O astfel de funcționalitate este utilă în cazul în care nu utilizează etichete împreună cu câmpurile text care să descrie conținutul acestora.

³⁸ Un obiect de tip `ScrollPane` poate conține un singur nod. Dacă se dorește 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 mișcarea mouse-ului.

³⁹ Implicit, proprietățile `FIT_TO_WIDTH` și `FIT_TO_HEIGHT` au valoarea false. De asemenea, clasa `ScrollPane` oferă posibilitatea de a se afișa dimensiunea minimă, dimensiunea maximă precum și dimensiunea curentă a componentelor pe care le conține.

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 metodele `setItems` (se folosește constructorul `FXCollections.observableArrayList`), respectiv `setCellFactory`. Modificarea dimensiunilor implicite se face prin `setPrefHeight`, respectiv `setPrefWidth`, cu valorile exprimate în pixeli. Lista poate fi orientată pe verticală sau orizontală, în funcție de parametrul pe care îl primește metoda `setOrientation` (`Orientation.HORIZONTAL` sau `Orientation.VERTICAL`).

```
ListView<Object> listView = new ListView<>();
ObservableList<Object> entries = FXCollections.observableArrayList (...);
listView.setItems(entries);
listView.setPrefWidth(...);
listView.setPrefHeight(...);
listView.setOrientation(...);
```

Clasele `SelectionModel` și `FocusModel` sunt folosite pentru a identifica selecția și focusul din cadrul listei⁴⁰:

- `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`⁴¹.

Dacă se dorește ca datele din listă să poată fi modificate de utilizator⁴², se pot utiliza extensii ale clasei `ListCell` ca `CheckBoxListCell`, `ChoiceBoxListCell`, `ComboBoxListCell` sau `TextFieldListCell`. Redefinirea implementării unei celule a listei la aceste tipuri de obiecte se face prin metoda `setCellFactory`.

JavaFX pune la dispoziție programatorilor și clase pentru a reprezenta datele într-o formă tabelară: `TableView`, `TableColumn` și `TableCell`. Popularea implică definirea unui model pentru tabel și asocierea unei fabrici pentru celule. Modelul de date este o clasă ale cărei atribute au tipul `SimpleStringProperty` și pentru care se definesc obiecte de tip getter și setter. Pentru fiecare coloană existentă în cadrul tabelului se va specifica, pe lângă nume și dimensiune și 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 Aceste proprietăți pot fi obținute, însă nu pot fi specificate pentru a determina valorile selectate, respectiv focusate în cadrul listei.

⁴¹ Obiectele `ListView` sunt create în mod implicit având ca model de selecție o implementare a clasei abstracte `MultipleSelectionModel`, cu proprietatea `selectionMode` având valoarea `SelectionMode.SINGLE`.

⁴² În mod implicit, datele dintr-un obiect de tipul `ListView` nu pot fi editate de utilizator.

Metoda `setCellValueFactory` specifică o fabrică de celule pentru fiecare coloană în parte. Aceasta este implementată folosind clasa `PropertyValueFactory` ce utilizează denumirile coloanelor tabelelor ca referințe către metodele aferente ale clasei `Entity`.

Tabelele JavaFX au capacitatea de a sorta datele⁴³ și de a redimensiona coloanele în funcție de conținut. În cazul în care se dorește 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	anapantie	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

Obiect de tip `TableView` conținând tabela „cărți” din aplicația `BookStore`

De asemenea, se pot crea structuri mai complexe, cum ar fi de exemplu coloane imbricate, adăugând la coloana părinte coloanele copil:

```
TableColumn child1 = new TableColumn("Child1");
TableColumn child2 = new TableColumn("Child2");
parent.getColumn().addAll(child1, child2);
```

În situația în care tabelul nu conține nici un fel de date, este afișat mesajul „No content in table”. Dacă se dorește afișarea altui text, acesta poate fi precizat prin metoda `setPlaceholder`.

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 `setRoot`). Un obiect `TreeView` poate conține elemente cu tipuri diferite. Pentru fiecare obiect de tip `TreeItem` poate fi asociat un element grafic ce poate fi precizat fie în constructor sau poate fi transmis ca parametru al metodei `setGraphic`. Acesta poate conține obiecte de orice tip (inclusiv alte controale), motiv pentru care clasa asociată este parametrizată `TreeItem<T>` (`T value`).

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 primește 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 dorește vizualizarea conținutului său.

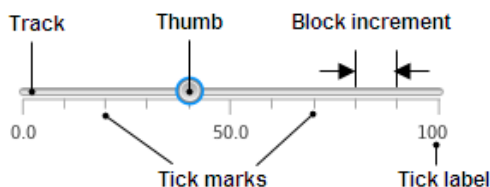
⁴⁵ Obiectele `TreeItem` pot fi expandate (vizualizându-se conținutul lor) sau nu, în funcție de parametrul metodei `setExpanded`. Implicit, conținutul obiectelor `TreeItem` nu poate fi vizualizat.

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. Acest comportament este util mai ales când obiectul respectiv va conține o cantitate mare de informații care este modificată în mod dinamic. De asemenea, o astfel de abordare poate fi folosită atunci când se dorește redefinirea tipului de control conținut, cu specificarea comportamentului aplicației în cazul când utilizatorul interacționează cu tipul de nod respectiv, putând fi folosite `CheckBoxTreeCell`, `ChoiceBoxTreeCell`, `ComboBoxTreeCell`, `TextFieldTreeCell`. Verificarea relațiilor dintre obiectele de tip `TreeItem` se face prin metodele `getParent`, respectiv `isLeaf`.

Obiectele de tip `ComboBox` sunt folosite atunci când este necesară selectarea unei opțiuni dintr-o listă care depășește o anumită limită, pentru că oferă posibilitatea vizualizării folosind elemente derulante, spre deosebire de obiectele `ChoiceBox`. Elementele conținute (obiect de tip listă observabilă) sunt specificate fie la construirea obiectului, fie apelând `setItems()`, 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 afișate. Ca și alte controale JavaFX, elementele de tip `ComboBox` pot fi editate (`setEditable(true)`), putându-se specifica și un text explicativ (`setPromptText()`).

Elementul `Separator` este utilizat doar pentru a distinge între elementele unui grup sau pentru a structura interfața grafică, fără a produce vreo acțiune. Fiind derivat din clasa `Node`, el poate fi stilizat, i se pot adăuga efecte vizuale și 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 poziția sa (orizontală sau verticală) se face cu `setValignment`, respectiv `setHalignment`, altfel implicit acesta ocupă tot spațiul care îi este alocat.

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



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 aceeași semnificație (`setMin`, `setMax`, `setValue`, `setShowTickLabels`, `setShowTickMarks`⁴⁷, `setMajorTickUnit`, `setMinorTickCount`, `setBlockIncrement`).

⁴⁶ Metoda `setValue()` modifică și obiectul asociat din proprietatea `selectionModel`.

⁴⁷ În cazul în care se dorește alinierea cursorului cu marcarele din cadrul pistei, se poate folosi metoda `setSnapToTicks`.

Metodele `setMin` și `setMax` definesc valorile extreme între care poate fi derulat obiectul de tip `Slider`, în timp ce `setValue` indică valoarea curentă a cursorului (care trebuie să fie mai mare decât valoarea minimă și mai mică decât valoarea maximă). Modul de vizualizare al obiectului `Slider` va fi definit de metodele `setShowTickMarks` și `setShowTickLabels` care primesc ca argumente parametrii de tip boolean. De asemenea, metoda `setBlockIncrement` definește distanța cu care cursorul se mișcă cât un utilizator apasă pe pista asociată. Valoarea la care se găsește cursorul poate fi obținută prin metoda `getValue` și este de obicei de tip `double`.

Prin clasele `ProgressIndicator` și `ProgressBar`⁴⁸ se indică utilizatorului faptul că o sarcină este în curs de execuție, precizându-se și 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 primește ca parametru procentul în care sarcina a fost completată⁴⁹ (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 cauză⁵⁰. Clasele mai pun la dispoziția programatorilor metoda `progressProperty` care întoarce valoarea progresului respectiv și pe care este aplicat obiectul ascultător în cazul modificărilor acesteia.

Clasa `Hyperlink`, derivată din `Labeled` (din care moștenește metodele `setText` și `setGraphic`) este folosită spre a formata texte care au semnificația legăturilor Internet. Starea unei legături (vizitată / nevizitată) poate fi marcată prin metoda `setVisited`. Acțiunea asociată operației de apăsare poate fi precizată prin metoda `setOnAction` care primește un obiect de tip `EventHandler`. La accesarea unui astfel de obiect, locația poate fi vizualizată printr-un obiect `WebEngine` (obținut ca `new WebView().getEngine()`) pentru care se apelează metoda `load`.

```
hiperlink.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        new WebView().getEngine().load(url);  
    }  
});
```

Obiectele `Tooltip` sunt folosite pentru texte explicative ce pot fi asociate oricăror controale JavaFX din pachetul `javafx.scene.control` prin apelul metodei `setTooltip`. Starea sa poate fi activat și vizibil⁵¹, de regulă existând o întârziere între cele două momente (plasarea mouse-ului asupra obiectului și afișarea propriu-zisă). Fiind derivată din `Labeled`, i se poate asocia nu numai un text, ci și o imagine. Legătura dintre un control de tip `Node` și obiectul `Tooltip` asociat poate fi eliminată prin metoda `uninstall` care primește ca argumente elementele care se doresc decuplate.

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

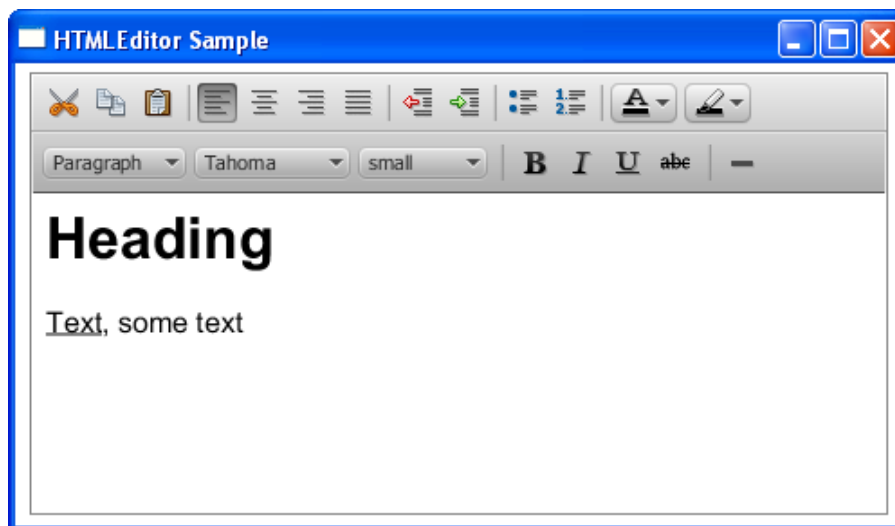
⁴⁹ Parametrul va avea o valoare subunitară.

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

⁵¹ Un obiect `Tooltip` care este vizibil este în același timp și activat.

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

- formatare de text (bold, italic, underline, strike through);
- configurări despre paragraf (formatare, tip, dimensiunea caracterelor);
- culori de fundal;
- indentare de text;
- liste numerotate și nenumerate;
- aliniere de text;
- folosirea unei rigle orizontale;
- facilități copy/paste pentru fragmente de text.



Un control de tip `HTMLEditor`

În cadrul editorului este prezentat conținutul paginii Internet asociat unui cod HTML. Constructorul său nu primește parametri, dar poate fi specificată dimensiunea prin metodele `setPrefWidth` / `setPrefHeight`. Conținutul său (o pagină Internet dată prin codul HTML) va fi specificat prin metoda `setHtmlText`. Acesta poate fi obținut folosind metoda „pereche” `getHtmlText`. Fiind derivat din clasa `Node`, acestui tip de obiect i se pot aplica stiluri, efecte vizuale sau transformări. Vizualizarea conținutului acestui editor într-un navigator poate fi realizată tot prin metoda `load` a unui obiect `WebEngine`, ca în cazul obiectul `Hyperlink`.

Obiectele de tip `TitledPane` reprezintă panouri care au asociat și un titlu, putând fi deschise și 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 afișate împreună. Obiectele `TitledPane` pot fi construite primind ca parametrii textul / obiectul de tip `Node`, acestea putând fi specificate ulterior prin metodele `setText`, respectiv `setContent`⁵². Implicit, orice obiect `TitledPane` poate fi închis și deschis (proprietate ce poate fi specificată prin metoda `setCollapsible`), acțiunea realizându-se în mod animat (metoda `setAnimated` indică acest comportament).

⁵² 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. Dimensiunea obiectului este ajustată astfel încât să poată fi vizualizat întreg conținutul său.

Un obiect `Accordion` conține mai multe obiecte `TitledPane` cu proprietatea că unul singur dintre acestea poate fi vizualizat la un moment dat⁵³. Obiectul de tip ascultător este aplicat obiectului întors de metoda `expandedPaneProperty` și este de tipul `ChangeListener`, parametrizat la tipul `TitledPane`.

```
Accordion accordion = new Accordion();
TitledPane titledPanel1 = new TitledPane(), titledPane2 = new TitledPane();
// ...
titledPanel1.setContent(...);
titledPane2.setContent(...);
accordion.getPanes().add(titledPanel1);
accordion.getPanes().add(titledPane2);
accordion.expandedPaneProperty().addListener(
    new ChangeListener<TitledPane>() {
        public void changed(ObservableValue<? extends TitledPane> oValue,
                           TitledPane oldValue, TitledPane newValue) {
            // ...
        }
    }
);
```

În JavaFX pot fi definite mai multe tipuri de meniuri⁵⁴ (`MenuBar`, `MenuItem` – `Menu`, `CheckMenuItem`, `RadioMenuItem`, `CustomMenuItem` (`SeparatorMenuItem`) ca și `ContextMenu`). Se construiește obiectul `MenuBar`, se definesc categoriile care sunt populate cu elemente de tip `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, fiind redimensionat la spațiul disponibil. 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 același efect ca și accesarea propriu-zisă a acestuia (folosind mouse-ul), apelând metoda `setAccelerator`, primind un parametru obținut din metoda statică `keyCombination` a clasei omonime (combinația are forma `Ctrl+...`, `Alt+...`, `Shift+...`).

⁵³ Obiectul `TitledPane` vizualizat în mod curent este accesat prin `{set/get}ExpandedPane`.

⁵⁴ Un meniu se definește ca o listă de elemente acționabile ce sunt afișate 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 și un mod de a economisi spațiul disponibil, plasându-se în cadrul meniurilor funcționalitatea care nu trebuie să fie vizibilă permanent.

O intrare de tipul `MenuItem` a unui meniu poate fi dezactivată⁵⁵ printr-un apel `setDisable(true)`.

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

Obiectele `RadioMenuItem` sunt de obicei asociate unui obiect `ToggleGroup` (prin metoda `setToggleGroup`) care implică selecția lor mutual exclusivă.

Obiectele `ContextMenu` sunt afișate atunci când este apelată metoda `show` asociată, de regulă la accesarea unui control (printr-un click de mouse⁵⁶) căruia i-a fost alocat meniul.

Obiectele `CustomMenuItem` sunt folosite atunci când intrările meniului utilizează alte tipuri de controale, derivate din clasa `Node`.

Controlul `ColorPicker` este un element de interfață cu utilizatorul care permite selecția unei culori dintr-o anumită gamă dar și specificarea acesteia⁵⁷ prin parametrii RGB (Red/Green/Blue) sau HSB (Hue/Saturation/Brightness). Obiectul conține o listă pentru selecția culorii, o paletă de culori precum și fereastra de dialog pentru specificarea parametrilor acesteia⁵⁸. Obiectul se poate crea fără parametri sau poate fi indicată o anumită culoare considerată curentă⁵⁹ (aceasta poate fi gestionată și 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>`). Culorile definite de utilizator nu pot fi reîncărcate la repornirea aplicației decât în situația în care este implementat un mecanism de persistență. Implicit, aspectul unui astfel de obiect este cel de `ComboBox`. Alternativ, pot fi precizate stiluri asemănătoare obiectelor `ArrowButton` sau `SplitButton`, folosind clasele CSS corespunzătoare, sau poate fi redefinită proprietatea `-fx-skin` a clasei CSS `.color-picker`⁶⁰.

```
colorPicker.getStyleClass().add("arrow-button");  
colorPicker.getStyleClass().add("split-button");
```

Clasa `Pagination` oferă posibilitatea de a naviga între mai multe pagini corespunzând unui conținut împărțit în mai multe secțiuni, fapt ce este util mai ales pentru dispozitivele încorporate unde suprafața de afișare este limitată, iar interfața grafică nu poate fi de cele mai multe ori încadrată în aceasta.

⁵⁵ O intrare dezactivată a unui meniu este continuare vizibilă, fără ca utilizatorul să aibă posibilitatea de a o selecta.

⁵⁶ Afișarea meniului contextual se face la coordonatele la care a fost înregistrat click-ul pe mouse.

⁵⁷ Utilizatorii își pot defini o culoare și prin mișcarea mouse-ului asupra suprafeței de culoare și asupra barei de culori, parametrii RGB/HSB fiind modificați în mod automat. De asemenea, culoarea poate fi modificată și indicând valoarea acesteia în format web (#, urmat de 6 cifre hexa). Transparența culorii poate fi specificată prin intermediul parametrul `Opacity`, care poate avea valori cuprinse între 1 și 100.

⁵⁸ Fereastra de dialog pentru specificarea parametrilor unei culori este deschisă în mod automat atunci când este selectată o anumită valoare din paleta de culori sau atunci când se selectează opțiunea „Custom Color...”.

⁵⁹ Clasa `Color` pune la dispoziție mai multe culori predefinite care au nume predefinite sau, pentru crearea de culori predefinite pot fi folosite metodele statice `rgb/hsb` (care primesc cei 3 parametri sau opțional 4, dacă se dorește și precizarea transparenței) respectiv metoda statică `web` care primește un șir de caractere.

⁶⁰ Încărcarea unei foi de stil care să fie valabilă pentru întreaga scenă poate fi realizată prin metoda `scene.getStyleSheets().add("sceneStyleSheet.css")`.

Controlul este format din conținutul propriu-zis al paginii (redat în funcție de logica aplicației), dar și din zona de navigare⁶¹. Constructorul unui obiect de acest tip poate fi apelat și fără parametri (caz în care numărul total de pagini are valoarea `INDETERMINATE`), poate primi 1 parametru indicând numărul total de pagini sau 2 parametri, indicându-se și numărul paginii selectate⁶². 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`⁶³.

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

În metoda de mai sus, metoda `call` întoarce un obiect de tip `Node` care reprezintă conținutul ce va fi atașat paginii în cauză. Se recomandă ca în cadrul acestei metode să se verifice faptul că parametrul cu care se apelează nu depășește numărul maxim de pagini.

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

Dacă se dorește ca în locul numărului paginii respective să se afișeze alt tip de indicator, se poate schimba tipul clasei CSS corespunzătoare⁶⁴. Atributele care pot fi precizate pentru a modifica stilul sunt:

- `-fx-max-page-indicator-count` – specifică numărul maxim de indicatori de pagină;
- `-fx-arrows-visible` – indică vizibilitatea săgeților care fac legătura cu paginile anterioară și următoare; în mod implicit, are valoarea `true`;
- `-fx-tooltip-visible` – precizează vizibilitatea informației explicative asociat indicatorului de pagină; în mod implicit, are valoarea `true`;
- `-fx-page-information-visible` – setează vizibilitatea informațiilor referitoare la pagină; în mod implicit, are valoarea `true`;
- `-fx-page-information-alignment` – reprezintă modul în care va fi aliniată secțiunea corespunzătoare informațiilor cu privire la pagină.

Controlul de tip `FileChooser` din pachetul `javafx.stage` permite utilizatorilor să navigheze prin sistemul de fișiere. El poate fi folosit pentru selectarea unui sau mai multor fișiere, respectiv pentru salvarea pe disc a unor anumite conținuturi.

⁶¹ În zona de navigare se găsesc butoane pentru deplasarea înainte și înapoi, butoane pentru fiecare pagină în parte (cel selectat fiind corespunzător paginii curente). Totodată, pagina curentă este specificată și printr-o etichetă având forma „pagina curentă / număr total de pagini”. Navigarea se face fie prin accesarea butoanelor înainte și înapoi, fie prin accesarea butonului corespunzător paginii în cauză.

⁶² În cazul specificării unei pagini curente, numerotarea începe întotdeauna de la 0. Alternativ, pot fi folosite metodele `setPageCount`, respectiv `setCurrentPageIndex`.

⁶³ Metoda de tip `callback` va fi apelată de fiecare dată când este selectată o pagină.

⁶⁴ Spre exemplu, în locul numerelor se pot folosi puncte, situație în care clasa CSS aferentă este `STYLE_CLASS_BULLET`.

Ulterior creării unui astfel de obiect (și asocierii unui titlu sugestiv), acesta poate fi afișat⁶⁵ în cadrul scenei respective în momentul în care este necesar, potrivit interacțiunii cu utilizatorul:

```
FileChooser fileChooser = new FileChooser();  
fileChooser.setTitle("Open File");  
// ...  
fileChooser.showOpenDialog(stage);
```

```
FileChooser fileChooser = new FileChooser();  
fileChooser.setTitle("Open Files");  
// ...  
fileChooser.showOpenMultipleDialog(stage);
```

Obiectul de tip `FileChooser` poate fi configurat și în privința directorului cu care se deschide, prin apelarea metodei `setInitialDirectory` care primește ca parametru un obiect de tip `File`⁶⁶.

Metoda `showOpenDialog` întoarce o valoare care specifică fișierul care a fost selectat de către utilizator sau `null` în cazul în care nu s-a realizat nici o selecție. Această metodă este folosită în situația în care se dorește să se poată selecta un singur fișier. Spre a se putea selecta mai multe fișiere concomitent, trebuie apelată metoda `showOpenMultipleDialog`; aceasta va întoarce o listă⁶⁷ a fișierelor care au fost selectate de utilizator sau `null` dacă nu s-a realizat nici o selecție. Astfel, aceste metode sunt blocante și nu întorc nici un rezultat până când utilizatorul nu furnizează un răspuns. Vizualizarea conținutului unui fișier (folosind aplicația asociată în mod implicit) se face prin clasa `java.awt.Desktop`⁶⁸ care pune la dispoziție metoda `open`⁶⁹.

Unui astfel de obiect i se pot aplica filtre cu privire la tipul fișierelor care vor fi afișate. Aceasta se face adăugând obiecte de tip `ExtensionFilter`⁷⁰ extensiilor deja existente la obiectul de tip `FileChooser` (lista acestora putând fi obținută prin metoda `getExtensionFilters()`).

Dacă se dorește salvarea unui conținut sub forma unui fișier pe disc, obiectul de tip `FileChooser` trebuie apelat cu metoda `showSaveDialog`. Acesta întoarce tot un obiect de tip `File` care poate fi folosit la scrierea unui conținut.

În situația în care un control predefinit nu corespunde necesităților, se pot crea controale definite de utilizator ce extind clasa `javafx.scene.control.Control`. De asemenea, comportamentul implicit al unor controale predefinite poate fi suprascris (chiar la crearea sa) redefinind metodele al căror mod de operare nu corespunde cerințelor. Pentru obiecte `TableView`, `ListView`, `TreeView`, `ComboBox` se pot defini „fabrici” de celule pentru generarea conținutului acestora.

⁶⁵ Look and feel-ul acestui tip de control este corespunzător sistemului de operare pe care este apelat, pentru acele sisteme de operare care îl au implementat.

⁶⁶ În cazul în care se dorește să se lase utilizatorului posibilitatea de a preciza directorul implicit, se poate folosi un obiect `DirectoryChooser`, iar rezultatul metodei `showDialog` (de tip `File`) să fie pasat metodei `setInitialDirectory` asociat obiectului de tip `FileChooser`.

⁶⁷ Această listă nu poate fi modificată. În cazul în care se încearcă acest lucru, va fi generată o excepție de tipul `UnsupportedOperationException`.

⁶⁸ Înainte de a folosi funcționalitățile puse la dispoziție de această clasă trebuie să se verifice dacă platforma pe care se rulează le suportă. În acest sens, se poate folosi metoda statică `isDesktopSupported`.

⁶⁹ Alte metode implementate de clasa `Desktop` sunt `browse` (pentru a deschide navigatorul implicit folosind o adresă dată ca parametru), `edit` (pentru editarea conținutului unui fișier), `mail` (pentru transmiterea unui mesaj electronic către o adresă specificată în câmpul `mailto`) și `print` (care tipărește la imprimanta implicită un fișier).

⁷⁰ Constructorul clasei `ExtensionFilter` primește ca parametri un șir de caractere sugestiv pentru tipul de fișiere respectiv și o listă de extensii în formatul `*.ext` (de exemplu `*.*`, `*.docx`, `*.pdf`, `*.jpg`, `*.png`).

În aplicațiile JavaFX componentele text pot fi create ca instanțe ale clasei `javafx.scene.text.Text` (derivată din clasa `Node`)⁷¹, respectiv `javafx.scene.text.TextBuilder`. În cazul obiectelor `Text`, la creare se poate specifica (sau nu) textul respectiv, precum și poziția unde se dorește a fi afișat. Pentru aceste obiecte se pot utiliza și metodele `setText`, `setFont`⁷² și `setColor`. Atunci când se folosesc obiecte de tip `TextBuilder` se folosește metoda statică `create`, apelându-se succesiv metoda `text` (pentru a specifica șirul de caractere respectiv) și apoi metoda `build`.

```
Text text = new Text();
text.setText("Sample Text");
text.setFont(Font.getDefault(), FontWeight.BOLD | FontPosture.ITALIC, 24);
text.setFontSmoothingType(FontSmoothingType.LCD);
text.setFill(Color.AZURE);
```

```
TextBuilder text = TextBuilder.create().text("Sample Text").build();
```

Se recomandă totuși ca astfel de proprietăți să fie precizate în foi de stil care apoi să fie încărcate prin intermediul metodei `setId` (care primește ca parametru numele clasei CSS corespunzătoare).

Efectele care se pot crea asupra textelor⁷³ sunt `PerspectiveTransform` (indicându-se cele 4 puncte ale formei ce definește transformarea de perspectivă ca perechi (x,y)), `GaussianBlur` pentru estomparea textului, `DropShadow` pentru umbre exterioare⁷⁴, `InnerShadow` pentru umbre interioare⁷⁵, `Reflection` pentru reflectarea conținutului respectiv⁷⁶. 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`⁷⁷.

```
Blend blend1 = new Blend(), blend2 = new Blend();
blend1.setMode(BlendMode.MULTIPLY);
blend2.setMode(BlendMode.MULTIPLY);
DropShadow dropShadow = new DropShadow(); //...
blend1.setBottomInput(dropShadow);
InnerShadow innerShadow = new InnerShadow(); // ...
Reflection reflection = new Reflection (); // ...
blend2.setBottomInput(innerShadow);
blend2.setTopInput(reflection);
blend1.setTopInput(blend2);
```

⁷¹ În acest mod se pot aplica efecte sau transformări și se pot crea animații folosind obiectul `text` respectiv. De asemenea, având în vedere că `Node` extinde `Shape`, se poate specifica și o culoare pentru fundalul textului, respectiv se poate aplica un efect de îngroșare pentru forma asociată lui.

⁷² Parametrul unei astfel de metode se obține prin metoda statică `font` a clasei cu același nume, specificându-se numele tipului de caractere (ca `String`) și dimensiunea acestuia. De asemenea, proprietățile de tip `bold` / `italic` pot fi specificate ca parametri ai aceleiași metode, utilizându-se constantele `FontWeight.BOLD` respectiv `FontPosture.ITALIC`. Se pot utiliza și tipuri de caractere definite de utilizator (conținute în fișiere de tip `.ttf` sau `.otf` aflate în directorul proiectului), acestea putând fi încărcate prin `loadFont`.

⁷³ Aplicarea unui efect se face prin metoda `setEffect`, care primește ca parametru obiect de tipul transformării respective.

⁷⁴ Pentru un obiect de tip `DropShadow` se poate specifica un `offset` (`setOffsetX/ setOffsetY`), o culoare (`setColor`), raza umbrei (`setRadius`), împrăștierea (`setSpread`).

⁷⁵ Obiectul de tip `InnerShadow` suportă aceleași metode ca și obiectul de tip `DropShadow`.

⁷⁶ Pentru un obiect de tip `Reflection` pot fi specificați parametrii precum opacitatea de sus și de jos (`setTopOpacity/setBottomOpacity`), fracția din original care va fi vizibilă în reflecție (`setFraction`), un `offset` între original și reflecție (`setTopOffset`).

⁷⁷ Atunci când se dorește combinare a mai mult de 2 efecte, combinarea se va face între un efect propriu-zis și alt obiect de tip `Blend` care conține la rândul său o altă combinație.

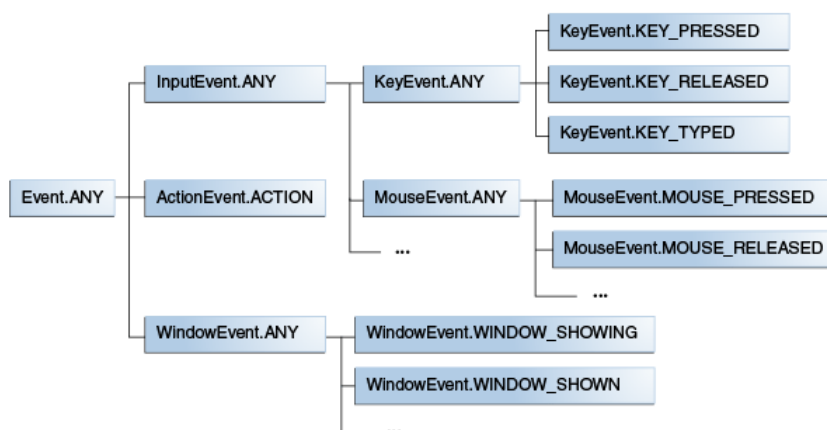
5. Tratarea evenimentelor asociate controalelor JavaFX

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 acesteia⁷⁸.

Proprietățile unui eveniment

Proprietate	Descriere
tip eveniment	tipul evenimentului care s-a produs
sursă	originea evenimentului, raportat la locația evenimentului în lanțul de propagare ⁷⁹
țintă	nodul asupra căruia a avut loc acțiunea și nodul frunză în lanțul de propagare ⁸⁰

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`⁸¹.



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.

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`⁸².

⁷⁸ Pe lângă tipurile predefinite de evenimente (`DragEvent`, `KeyEvent`, `MouseEvent`, `ScrollEvent`), programatorul își poate defini propriile tipuri de evenimente (care vor extinde clasa `Event`).

⁷⁹ Sursa se modifică pe măsură ce evenimentul este transmis de-a lungul lanțului de propagare.

⁸⁰ Deși ținta nu se schimbă, dacă un filtru pentru evenimente consumă evenimentul în procesul de tratare a evenimentului, ținta nu va primi evenimentul.

⁸¹ 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.

⁸² Clasele `Window`, `Scene` și `Node` implementează interfața `EventTarget` și toate subclasele lor moștenesc 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` și `Node`, pentru acestea va trebui să se implementeze interfața `EventTarget`, pentru ca acesta să gestioneze acțiunile utilizatorului.

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 și 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 respectiv⁸³;
- 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 apăsări⁸⁴.

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 folosește lanțul de propagare a evenimentului dat de metoda `buildEventDispatchChain()` de la nivelul nodului țintă selectat (aceasta va conține în mod obligatoriu obiectele `Stage` și `Scene` care conțin nodul respectiv și apoi calea propriu-zisă către nodul respectiv în graful de scene⁸⁵).

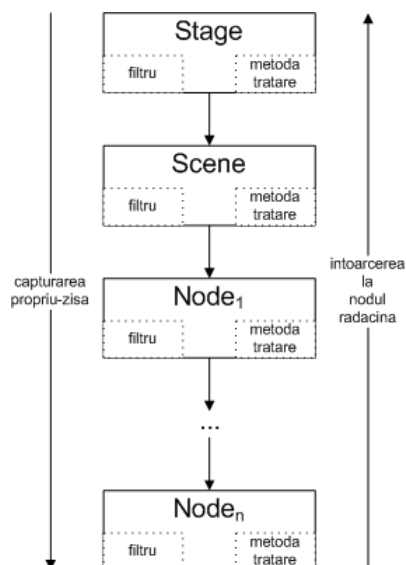
În etapa de *capturare propriu-zisă*, evenimentul în cauză este transmis de la nodul rădăcină către nodul țintă prin lanțul de propagare a evenimentului. Fiecare nod din acest lanț de propagare al evenimentului care are implementată o metodă pentru respectivul tip de eveniment îl va gestiona, ulterior 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.

După ce evenimentul a ajuns la nodul țintă și / 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 pentru respectivul tip de eveniment, aceasta este apelată, transmițând evenimentul mai departe după ce se termină. Evenimentul va fi transmis într-un final și nodului rădăcină doar dacă metodele de tratare a acestuia nu îl consumă.

⁸³ 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.

⁸⁴ 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.

⁸⁵ Totuși, 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).



Mecanismul de filtrare și tratare al evenimentelor în JavaFX

În JavaFX, gestiunea evenimentelor se face prin filtre și 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 respectiv⁸⁶.

Un *filtru* este executat în etapa de capturare propriu-zisă. La nivelul unui nod părinte, pot fi implementate filtre pentru mai multe noduri fiu și 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 evenimente⁸⁷.

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. Și în cazul metodelor de tratare se aplică aceeași regulă din cazul filtrelor cu privire la numărul de evenimente pe care acestea îl pot gestiona precum și a ordinii în care acestea sunt executate⁸⁸.

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.

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 își au definite propriile mecanisme de oportunitate.

⁸⁶ Diferența dintre un filtru și o metodă de tratare constă în momentul la care fiecare dintre acestea sunt executate.

⁸⁷ 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 același nivel.

⁸⁸ Evenimentele specificate prin metode de oportunitate (*eng. convenience methods*) se execută ultimele.

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 mișcat 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, mișcarea mouse-ului urmată de eliberarea butonului.	MouseEvent	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 și 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 întâlnește 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 e închisă, vizibilă, ascunsă	WindowEvent	Window

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

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

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

event-class este clasa care definește tipul de eveniment⁸⁹.

⁸⁹ Sintaxa <? super event-class> indică faptul că metoda acceptă o metodă de tratare fie pentru clasa care definește tipul de eveniment (event-class), fie pentru una dintre superclasele acesteia.

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) {  
        // ...  
    }  
});
```

Eliminarea metodei de tratare a evenimentului se face prin specificarea unui parametru cu valoarea null pentru nodul în cauză, dacă se dorește ca acesta să nu mai proceseze astfel de evenimente.

```
tableContent.setOnMouseClicked(null);
```



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 să 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 și filtrul propriu-zis. Eliminarea unui filtru de eveniment corespunzător unui nod se face prin metoda `removeEventFilter`, primind aceiași 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);
```

De observat faptul că filtrul este definit pentru un tip de eveniment generic (`InputEvent`), fiind utilizat apoi pentru tratarea unor evenimente specifice (legate de mouse și de tastatură).

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 și 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 să 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 și filtrul propriu-zis. Eliminarea unei metode de tratare corespunzătoare unui nod se face prin metoda `removeEventHandler`, primind aceiași 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);
```

Și în acest caz metoda de tratare este definită pentru un tip de eveniment generic urmând a fi folosită pentru tratarea unor evenimente specifice.

Pentru anumite tipuri de dispozitive încorporate, pot fi detectate evenimente de tipul atingerilor, operațiilor de micșorare/mărire (*eng.* zoom), rotire, derulare (*eng.* scroll) sau parcurgere (*eng.* swipe). Aceste tipuri de gesturi pot implica unul sau mai multe puncte de contact. Tipul de eveniment generat corespunde gestului pe care îl realizează utilizatorul.

Gesturi suportate și tipurile de evenimente asociate

Gest	Descriere	Evenimente generate
Rotație	Mișcarea unui deget în jurul altuia în sensul acelor de ceasornic sau în sens contrar acelor de ceasornic	ROTATION_STARTED ROTATE ROTATION_FINISHED
Derulare	Mișcare de derulare, pe verticală (sus/jos) sau pe orizontală (stânga/dreapta)	SCROLL_STARTED SCROLL SCROLL_FINISHED Dacă se folosește roțița de mouse se generează doar evenimentul SCROLL.
Parcurgere	Mișcare de parcurgere de-a lungul ecranului la stânga, dreapta, în sus sau în jos. Mișcarea pe diagonală nu este recunoscută ca parcurgere.	SWIPE_LEFT, SWIPE_RIGHT SWIPE_UP, SWIPE_DOWN Pentru fiecare gest se generează un singur eveniment de parcurgere.
Mărire/Micșoare	Depărtarea a două degete pentru mărire, respectiv apropierea lor pentru micșorare.	ZOOM_STARTED ZOOM ZOOM_FINISHED

Așa cum s-a precizat, ținta celor mai multe gesturi este reprezentată de nodul aflat în centrul tuturor punctelor de apăsare de la începutul gestului respectiv⁹⁰.

Unele gesturi pot genera și alte tipuri de evenimente în plus față de cele care erau intenționate⁹¹. Pentru ca același eveniment să nu fie procesat de 2 ori, trebuie apelată metoda `isSynthesized` pentru a se verifica dacă evenimentele nu au fost generate din alte evenimente. În același scop poate fi folosită metoda `isDirect` care indică faptul că evenimentul a fost generat de un gest realizat pe ecranul tactil. Pentru a verifica dacă anumite evenimente nu s-au generat ca urmare a inerției unui gest se poate apela metoda `isInertia`.

În cazul în care se dorește ca evenimentele de apăsare a ecranului să fie tratate individual (sau se dorește implementarea de alte gesturi) pot fi utilizate evenimentele puse la dispoziție de clasa `TouchEvent`, ce pot avea tipurile `TOUCH_PRESSED`, `TOUCH_MOVED`, `TOUCH_STATIONARY` și `TOUCH_RELEASED`. Pentru fiecare apăsare a ecranului (punct de contact⁹²) se poate genera un eveniment având acest tip⁹³. Un eveniment de apăsare are asociate următoarele elemente:

- punctul de contact – reprezintă punctul principal de contact asociat evenimentului respectiv;
- numărul de apăsări – numărul punctelor de contact legate de acțiunea de apăsare;
- lista punctelor de contact – mulțimea punctelor de contact asociate evenimentului de apăsare în cauză;
- identificatorul mulțimii de evenimente – identificatorul mulțimii ce conține evenimentul de apăsare⁹⁴.

Nodul țintă al unui eveniment de apăsare corespunde cu nodul țintă al punctului de contact, reprezentând controlul cel mai vizibil (de pe nivelul frunză al grafului de scene) de la coordonatele respective. Toate evenimentele asociate unui punct de contact sunt transmise aceluiași nod țintă. Totuși un nod care procesează un eveniment poate apela metoda `grab()` pentru a deveni ținta respectivului punct de contact⁹⁵, sau poate specifica această țintă ca parametru al metodei respective: `grab(target)`. De asemenea, metoda `ungrab` este folosită pentru a detașa punctul de contact de ținta curentă, evenimentele asociate fiind transmise nodului corespunzător locației la care se află punctul de contact.

⁹⁰ În situația în care există mai mult de un nod aflat la poziția respectivă, ținta este considerată nodul aflat deasupra celorlalte.

⁹¹ Operațiile de parcurgere pot genera operații de derulare în funcție de amplitudinea gestului. Totodată, operațiile de atingere generează atât evenimente de tip `TOUCH_PRESSED` / `TOUCH_RELEASED` cât și evenimente de tip `MOUSE_PRESSED` / `MOUSE_RELEASED`. În aceste cazuri, este posibil ca nodurile țintă asociate celor două evenimente să fie diferite. De asemenea, în acest mod, aplicațiile desktop pot fi portate foarte ușor pe sisteme încorporate ce utilizează ecranul tactil.

⁹² Un punct de contact este reprezentat de o instanță a clasei `TouchPoint` conținând informații despre locație, stare și nodul țintă al punctului de contact.

⁹³ În cazul apăsărilor concomitente ale ecranului tactil, numărul de evenimente generate poate fi limitat de platforma pe care rulează aplicația.

⁹⁴ Fiecare set de evenimente are un identificator unic care este incrementat pentru fiecare mulțime care este generată ca urmare a unei operații de apăsare. Evenimentele din mulțime pot avea tipuri diferite sau același tip. De aceea, fiecare punct de contact este identificat și el printr-un identificator unic.

⁹⁵ În acest fel, există posibilitatea ca toate evenimentele aparținând unei anumite mulțimi să fie tratate de același nod țintă

6. Utilizarea de elemente grafice avansate și conținut multimedia

JavaFX oferă programatorului posibilitatea de a crea animații, sub forma tranzițiilor sau a animațiilor bazate pe timp, implementate în clasele `Transition` și `Timeline` din pachetul `javafx.animation.Animation`.

Tranzițiile implementează animațiile asociindu-le o anumită cronologie (intern), acestea putând fi executate secvențial sau în paralel. În JavaFX, au fost implementate mai multe tipuri de tranziții [6]:

- **`FadeTransition`** – este folosită pentru a schimba opacitatea unui nod de-a lungul unei secvențe de timp; constructorul primește durata (exprimată în milisecunde) și nodul pe care se va realiza animația; ulterior se pot specifica valorile extreme ale transparenței (prin metodele `setFromValue` respectiv `setToValue`), dacă va fi reluată după terminarea ei (metoda `setAutoReverse`) și numărul de rulări (metoda `setCycleCount`); rularea animației se face prin metoda `play`;
- **`PathTransition`** – este folosită pentru a muta un nod de-a lungul unei căi într-o anumită perioadă de timp; pentru o astfel de animație se specifică, suplimentar, calea (prin metoda `setPath` care primește ca parametru un obiect de tip `Path`⁹⁶) precum și orientarea nodului respectiv de-a lungul căii (valorile posibile sunt exprimate în `PathTransition.OrientationType`);
- **`FillTransition`** – schimbă culoarea suprafeței de umplere a unui nod de-a lungul unui interval de timp; valorile limită trebuie să aibă semnificația unor culori;
- **`StrokeTransition`** – schimbă culoarea conturului unui nod de-a lungul unui interval de timp; valorile limită trebuie să aibă semnificația unei culori;
- **`PauseTransition`** – folosită ca perioadă de așteptare între tranziții, în cadrul unei tranziții secvențiale sau apelată în cadrul `Animation.onFinished`;
- **`RotateTransition`** – utilizată pentru rotirea unui nod cu un anumit unghi (dat în grade) de-a lungul unei perioade de timp;
- **`ScaleTransition`** – folosită pentru scalarea unui nod cu anumite valori pe axele `Ox`, `Oy`, `Oz`; pot fi folosiți atât factori de multiplicare cât și intervale de valori (pentru fiecare axă în parte);
- **`TranslateTransition`** – modifică poziția nodului între anumite coordonate $(x_1, y_1, z_1) \rightarrow (x_2, y_2, z_2)$ sau de la poziția curentă la o poziție obținută prin intermediul unor multiplicatori;
- **`ParallelTransition`** – utilizată spre a executa mai multe tranziții simultan; spre exemplu, dacă se definesc mai multe tranziții asupra unui nod, acestea pot fi adăugate unui obiect `ParallelTransition` prin metoda `addAll` aplicabilă listei copiilor acestuia (obținută cu metoda `getChildren`) pentru a fi executate în paralel; durata tranziției obținute va fi dată de durata celei mai lungi tranziții din cadrul listei asociate;
- **`SequentialTransition`** – funcționează similar obiectului `ParallelTransition` cu excepția faptului că tranzițiile membre sunt executate secvențial, în ordinea în care au fost adăugate, iar durata tranziției este dată de suma duratelor tranzițiilor componente.

⁹⁶ Un obiect de tip `Path` se construiește din obiecte de tip `CubicCurveTo` care definește o curbă Bézier între elementul curent și un anumit punct (x, y) trecând prin alte două puncte de control $(controlX1, controlY1)$ și $(controlX2, controlY2)$ date ca parametrii la construirea obiectului.

În cadrul animațiilor bazate pe timp, se consideră că nodul ce va fi animat este definit de anumite proprietăți care pot fi modificate automat de către sistem atâta vreme cât se precizează valorile sale extreme (de început și de sfârșit, numite cadre cheie – *eng.* key frames). De asemenea, astfel de obiecte (implementate de clasa `Timeline`) au posibilitatea de a fi oprite (cu sau fără posibilitatea reluării animației din punctul la care au fost oprite), de a fi redată în sens invers sau de a fi repetate.

Un obiect `KeyFrame` primește la creare ca parametru durata animației și un obiect (sau mai multe) de tip `KeyValue` care specifică un atribut ce se dorește modificat în intervalul respectiv. De asemenea, acesta poate asocia un eveniment (de tipul `EventHandler<ActionEvent>`, implementând metoda `handle`) care de regulă este generat la sfârșitul animației.

JavaFX oferă posibilitatea de a folosi obiecte de interpolare (fie existente, fie definite de utilizator), specificând diferite poziții intermediare ale obiectului pentru care se realizează animația. Un obiect `Interpolator` este specificat pentru un obiect `KeyValue` atunci când este creat⁹⁷. Tipurile de interpolare predefinite sunt `Interpolator.LINEAR`, `Interpolator.DISCRETE`, `Interpolator.EASE_BOTH`, `Interpolator.EASE_IN`, `Interpolator.EASE_OUT`. În cazul în care utilizatorul dorește să definească un obiect de tip interpolare, trebuie să definească o clasă (ce extinde `Interpolator`) în care trebuie să definească metoda `curve`, folosită pentru a determina fracția în implementarea metodei `interpolate`.

În JavaFX pot fi utilizate și grafice, fiind suportate următoarele tipuri: bar, area, line, bubble, scatter, pie [7].



Tipuri de grafice în JavaFX

⁹⁷ În cazul în care nu se specifică nici un obiect de tip `Interpolator`, va fi folosit în mod implicit valoarea `Interpolator.LINEAR`.

În momentul în care se definește un model pentru un anumit tip de grafic, trebuie să se distingă între acele tipuri de tabele care folosesc axele de coordonate și cele care nu le utilizează.

Graficele ce utilizează sistemul de coordonate (de tipul bar, area, line, bubble, scatter) sunt derivate din clasa `XYChart`, modelul de date fiind specificat de clasa `XYChart.Data`. Valorile aferente axelor de coordonate sunt reprezentate de proprietățile `xValue`, respectiv `yValue`. De asemenea, pentru fiecare element al graficului se pot specifica valori suplimentare. În cazul acestor tipuri de grafice pot fi definite mai multe serii de date⁹⁸, folosind clasa `XYChart.Series`. Pentru fiecare axă poate fi specificată eticheta, poziția relativă față de grafic (apelând metoda `setSide` pentru obiectul de tip `NumberAxis` / `CategoryAxis`), limitele inferioară și superioară (`setLowerBound` / `setUpperBound`), distanța dintre marcaje (`setTickUnit`), etichetele acestora și valorile între care vor fi poziționate. Totodată, axele își pot determina unele proprietăți din datele care le sunt asociate. În momentul în care sunt modificate proprietățile axelor de coordonate sau din cadrul graficului se poate specifica faptul că operația să se realizeze în mod animat, printr-un apel al metodei `setAnimated`. Dacă datele conținute în tabel sunt de tip numeric, axele cu care se crează acesta au tipul `NumberAxis`, altfel se va folosi `CategoryAxis`.

```
CategoryAxis xAxis = new CategoryAxis();
NumberAxis yAxis = new NumberAxis();
yAxis.setTickLabelFormatter(new NumberAxis.DefaultFormatter(yAxis, "RON ", null));
LineChart<String, Number> lineChart = new LineChart<String, Number>(xAxis, yAxis);
lineChart.setTitle("Profit Statistics");
XYChart.Series series2013 = new XYChart.Series();
series2013.setName("Year 2013");
series2013.getData().add(new XYChart.Data("January", 10000));
// ...
lineChart.getData().add(series2013);
```

Graficele care nu folosesc sistemul de coordonate (pie), derivate din clasa `PieChart`, folosesc `PieChart.Data` pentru a indica valorile fiecărei secțiuni. În acest tip de grafic, datele sunt afișate în ordinea acelor de ceasornic (modul de afișare putând fi schimbat prin metoda `setClockwise`) și au un anumit unghi de pornire (ce poate fi modificat prin metoda `setStartAngle`).

Proprietățile comune tuturor tipurilor de grafice sunt titlul (`setTitle`), poziția sa (sus, jos, stânga, dreapta), vizibilitatea etichetelor (`setLabelsVisible`), vizibilitatea (`setLegendVisible`) și poziția (`setLegendSide` / `setLabelLineLength`) legendei.

Un element al unui grafic poate avea asociată o metodă de tratare în cazul producerii unor evenimente, aceasta realizându-se similar ca pentru orice obiect din clasa `Node`. Metodele de tratare trebuie adăugate pentru fiecare nod în parte, obținut prin interare pe rezultatul metodei `chart.getData()`.

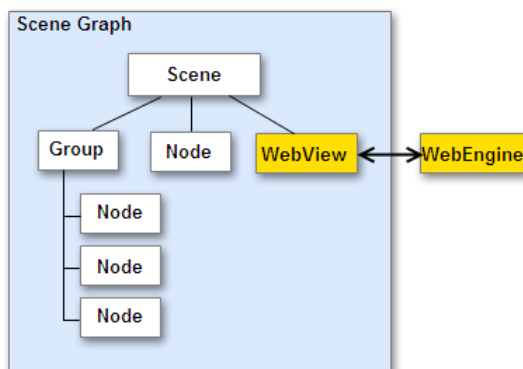
Și pentru un grafic poate fi schimbat modul de vizualizare prin asocierea unei foi de stil sau a unei clase CSS⁹⁹.

Utilizatorul își poate defini propriul tip de tabel (care extinde clasa `Chart`) și suprascriind metodele definite în această clasă.

⁹⁸ Fiecare serie de date poate avea un nume, iar valorile sunt adăugate prin metoda `add` apelabilă pe rezultatul metodei `XYChart.Series.getData()`, acestea având tipul `XYChart.Data`.

⁹⁹ Proprietățile CSS corespunzătoare sunt prefixate de regula de `.default-colorx` (cu `x` cuprins între 0 și 2) și pot avea valorile `chart-area-symbol`, `chart-series-area-line`, `chart-series-area-fill`.

JavaFX dispune și de un navigator încorporat, bazat pe WebKit, care implementează o funcționalitate completă ca fi redarea de conținut HTML5¹⁰⁰ stocat local sau la o adresă la distanță, obținerea istoricului de vizitare, executarea de comenzi JavaScript și de apel al metodelor JavaFX din acest context, gestiunea ferestrelor, supotând de asemenea foile de stil (CSS) și interfața de programare DOM (Document Object Model). Acesta moștenește toate atributele și metodele de la clasa Node, având toate funcționalitățile acesteia. Clasele pe care le folosește navigatorul se găsesc în pachetul `javafx.scene.web`.



Arhitectura navigatorului încorporat [8]

Clasa `WebEngine` oferă funcționalitate de navigare de bază¹⁰¹ implementând interacțiunea cu utilizatorul cum ar fi accesarea unor legături Internet și transmiterea de formulare HTML. O instanță a acestei clase poate fi creată folosind un constructor fără parametri (conținutul Internet aferent putând fi indicat prin metoda `load`) sau un constructor căruia i se transmite o locație.

Clasa `WebView` reprezintă o extensie a clasei `Node`, încapsulând obiectul `WebEngine`, încorporând conținutul HTML în scena unei aplicații și oferind atributele și metodele pentru a aplica efecte și transformări. Legătura dintre acest obiect și motorul asociat se face prin metodele `{get|set}Engine`.

```
WebView webView = new WebView();
WebEngine webEngine = webView.getEngine();
webEngine.load("http://cs.curs.pub.ro");
```

Modul în care sunt gestionate alte ferestre având alt conținut (*eng.* pop-up windows) poate fi specificat prin intermediul unei metode de tratare care poate întoarce același motor web ca cel curent (caz în care pagina respectivă este deschisă în același navigator) sau alt motor web:

```
webEngine.setCreatePopupHandler(new Callbak<PopupFeatures, WebEngine>() {
    @Override
    public WebEngine call(PopupFeatures config){
        // ...
    }
});
```

¹⁰⁰ În prezent, JavaFX suportă următoarele caracteristici HTML5: Canvas, redarea de conținut media, controale de tip formular (cu excepția `<input type="color">`), conținut editabil, menținerea istoricului, etichetele `<meter>`, `<progress>`, `<details>`, `<summary>`, API-ul DOM și formatul SVG, nume de domenii scrise în limbă națională.

¹⁰¹ Clasa `WebEngine` gestionează o singură pagină HTML la un moment dat, oferind funcționalități de navigare de bază cum ar fi încărcarea de conținut Internet și accesarea proprietăților DOM corespunzătoare precum și execuția unor comenzi JavaScript (această funcționalitate poate fi dezactivată pentru anumite motoare web).

Clasa `WebEngine` pune la dispoziție metoda `executeScript` pentru a rula o comandă JavaScript aferentă documentului încărcat în navigator la momentul de timp respectiv.

De asemenea, există posibilitatea de a invoca cod JavaFX din cadrul JavaScript prin crearea unui obiect de interfață în aplicație care este transmis prin metoda `JSObject.setMember()`. Ea trebuie apelată dintr-o metodă de tratare asociată evenimentului de încărcare al paginii, care se realizează pe un fir de execuție separat. Pentru acest obiect, atributele și metodele publice definite în JavaFX vor fi accesibile din JavaScript.

```
webEngine.getLoadWorker().stateProperty().addListener(  
    new ChangeListener<State>() {  
        @Override  
        public void changed(ObservableValue<? extends State> oValue,  
                            State oldState, State newState) {  
            if (newState == State.SUCCEEDED) {  
                JSObject jsObject = (JSObject)webEngine.executeScript("window");  
                jsObject.setMember("myObject", new MyClass());  
            }  
            // ...  
        }  
    });  
// ...  
public class MyClass {  
    public void myMethod() {  
        // ...  
    }  
}
```

Un astfel de obiect poate fi folosit într-un document HTML, astfel

```
<a href="about:blank" onclick="jsObject.myMethod()">...</a>
```

Pentru obiectele `WebView`, meniurile contextuale sunt activate implicit, dezactivarea lor realizându-se prin metoda `setContextMenuEnabled`.

Istoricul paginilor vizitate poate fi obținut prin metoda `getHistory` aplicată obiectului de tip `WebEngine`, aceasta întorcând un obiect de tip `WebHistory`. Acesta este o listă de intrări ce poate fi accesată prin metoda `getEnteries`, conținând informații despre pagină (URL și titlu) sau despre data la care aceasta a fost vizitată cel mai recent. Metoda `go` a acestei clase determină navigatorul să încarce pagina reținută la indexul respectiv.

În cazul în care se dorește folosirea mai multor navigatoare, acestea pot fi create în cadrul unor obiecte de tip `TabPane`.

JavaFX oferă suport și pentru operații de tip drag-and-drop, ce implică un transfer de date între două obiecte: sursa gestului și ținta gestului, acestea putând fi implementări ale obiectelor de tip `Node` și `Scene`, acestea putând aparține aceleiași aplicații sau unor aplicații diferite¹⁰². O astfel de operație este implementată folosind clasa `javafx.scene.input.DragEvent`. Modurile de transfer suportate între sursă și țintă sunt `COPY`, `MOVE` și `LINK`. Gestul de tip drag and drop poate fi pornit doar din cadrul unui nod sursă prin apelarea metodei `startDragAndDrop` care primește ca argument tipurile de transfer pe care aceasta le suportă¹⁰³, în cadrul metodei de tratare a evenimentului `DRAG_DETECTED`.

¹⁰² De asemenea, una dintre aplicații poate să nu fie aplicație JavaFX ci o aplicație a sistemului de operare.

¹⁰³ Pentru a specifica toate modurile de transfer, se va indica parametrul `TransferMode.ANY`.

Acceptarea unei operații se face în cadrul nodului țintă prin metoda `acceptTransferModes` în cadrul metodei de tratare a evenimentului `DRAG_OVER` prin specificarea tipurilor de transfer pe care le acceptă. De regulă, se verifică tipul de date pe care îl conține obiectul pentru care se dorește să se realizeze operația de transfer, conținut ce poate fi obținut prin metoda `getDragboard`. Trecerea obiectului pentru care se încearcă operația drag and drop deasupra anumitor noduri generează evenimente de tip `DRAG_ENTERED` sau `DRAG_EXITED`. Astfel, pot fi definite metode de tratare ale acestor evenimente care să schimbe aspectul acestui tip de obiecte pentru a indica faptul că nodul respectiv acceptă sau respinge transferul în cauză. Pe lângă conținutul obiectului se poate verifica și sursa acestuia prin metoda `getGestureSource()`. În momentul în care obiectul este transferat asupra țintei, aceasta primește un eveniment de tip `DRAG_DROPPED` în care pot fi preluate informațiile pe care le conține marcând rezultatul operației prin metoda `setDropCompleted` care primește un argument de tip boolean. Când gestul drag and drop este încheiat, sursa primește un eveniment `DRAG_DONE` care poate fi tratat în sensul că pentru anumite moduri de transfer trebuie realizate operații suplimentare¹⁰⁴.

JavaFX permite și transferul de date definite de utilizator însă în acest caz tipul de date trebuie să fie specificat explicit și trebuie să fie serializabil.

Pachetul `javafx.scene.media` oferă funcționalități pentru redarea de conținut multimedia în cadrul aplicației JavaFX.

Formatele suportate în mod curent sunt:

- **audio:** MP3; AIFF și WAV conținând PCM (Pulse Code Modulation) necomprimat; MPEG-4 folosind AAC (Advanced Audio Coding);
- **video:** FLV în care formatul video este VP6, iar cel audio este MP3; MPEG-4 cu compresie video H.264/AVC (Advanced Video Coding).

Decodificarea unor tipuri de compresii audio sau video¹⁰⁵ se bazează pe motoarele multimedia ale sistemelor de operare.

Printre funcționalitățile pe care le pune la dispoziție pachetul JavaFX pentru conținut multimedia se numără redarea tipurilor de formate amintite, suport pentru protocoalele `HTTP` și `FILE`, descărcarea de conținut în mod progresiv, căutare, folosirea unei zone de memorii tampon (*eng.* buffer) și funcții legate de redare (play, pause, stop, volume, mute, balance, equalizer). Totodată, poate fi redat conținut aflat în Internet, parametrii de redare (rata de biți pentru audio, rezoluția pentru video) fiind ajustați în funcție de condițiile existente. Capabilitățile multimedia sunt extinse la comutarea între fluxuri alternative, așa cum sunt specificate în cadrul listei de redare (*eng.* playlist)¹⁰⁶ în funcție de limitările impuse de rețea.

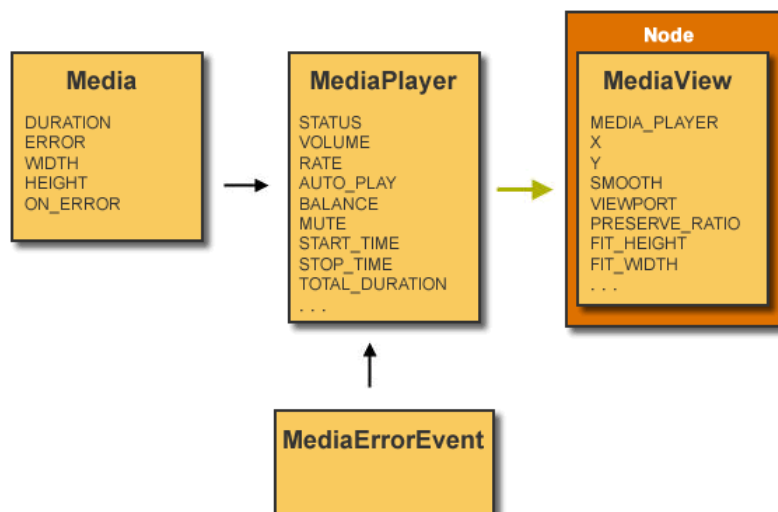
¹⁰⁴ În cazul operației dem mutare (`TransferMode.MOVE`) conținutul transferat trebuie eliminat din cadrul nodului sursă.

¹⁰⁵ Decodificarea unor formate AAC sau H.264/AVC au unele limitări dependente de platformă. JavaFX nu își propune să trateze toate formatele multimedia și tipurile de codificare din aceste containere, ci funcționalitate asemănătoare a acestor obiecte pe toate platformele pe care aceasta a fost implementată.

¹⁰⁶ Pentru un flux, există o listă de redare precum și o listă de segmente în care fluxul este corupt. Fluxul poate avea formatul MP3 sau MPEG-TS ce conține o multiplexare între conținut audio AAC și conținut video H.264.

Conceptul multimedia al JavaFX se bazează pe următoarele clase, care trebuie utilizate împreună pentru a implementa funcționalitatea de redare a conținuturilor de acest tip [9]:

- **Media** – o resursă multimedia, conținând informații (metadate) despre aceasta;
- **MediaPlayer** – componenta care oferă funcționalitățile pentru redarea conținutului multimedia respectiv;
- **MediaView** – un obiect `Node` care suportă animații și efecte.



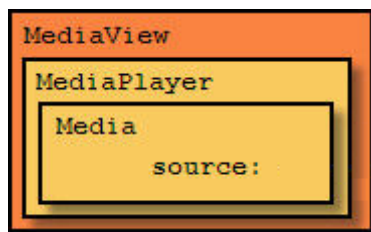
Clasele din pachetul `javafx.scene.media` [9]

Clasa `MediaPlayer` oferă toate atributele și metodele necesare pentru controlul redării de conținut multimedia. Astfel, se poate preciza proprietatea `AUTO_PLAY`, se poate apela direct metoda `play()` sau se poate specifica explicit numărul de repetări ale conținutului respectiv. Atributele `VOLUME` și `BALANCE` pot fi folosite pentru ajustarea nivelului de redare și a echilibrului stânga-dreapta¹⁰⁷. Alte metode ce pot fi apelate pentru redarea conținutului sunt `pause()` și `stop()`. Utilizatorul își poate defini mai multe metode de tratare pentru evenimente legate de starea obiectului care redă conținutul și anume: transferul fluxului care trebuie redat în zona tampon de memorie, terminarea conținutului, stagnare datorată faptului că datele nu au fost primite suficient de rapid spre a asigura redare continuă, întâlnirea de erori definite în clasa `MediaErrorEvent`.

Clasa `MediaView` extinde clasa `Node` și oferă o vizualizare a conținutului ce este redat, fiind responsabilă mai mult pentru efecte și transformări. Atributul `mediaPlayer` specifică obiectul `MediaPlayer` al cărui conținut este redat curent. Alte atribute sunt `onError`, indicând metoda de tratare care va fi invocată când se produce vreo eroare, `preserveRatio` care specifică faptul că raportul lățime/înălțime trebuie păstrat în cazul operațiilor de scalare, `smooth` pentru ca operația de scalare¹⁰⁸ să folosească un algoritm de filtrare mai performant, `viewport` pentru cadrul de vizualizare al conținutului multimedia, respectiv coordonatele `x` și `y` ale originii obiectului.

¹⁰⁷ Pentru atributul `VOLUME` se pot specifica valori cuprinse între 0.0 și 1.0, iar pentru `BALANCE` valori cuprinse între -1.0 (doar stânga), 0 (centru) și 1.0 (doar dreapta).

¹⁰⁸ Operația de scalare are loc în momentul în care conținutul multimedia trebuie ajustat la dimensiunile obiectului care îl conține, dimensiuni specificate de atributele `fitHeight` și `fitWidth`.



```
Scene scene = new Scene(new Group(), 500, 500);
String source = getParameters().getRaw().get(0);
Media media = new Media(source);
// Media media = new Media(MEDIA_URL);
MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.setAudio(true);
MediaView mediaView = new MediaView(mediaPlayer);
((Group)scene.getRoot()).getChildren().add(mediaView);
```

Unui astfel de obiect `MediaView` trebuie să i se adauge obiecte din categoria controalelor JavaFX astfel încât utilizatorul să aibă funcționalitatea pe care o oferă orice produs ce redă conținut multimedia (cronometru și bară de progres care indică momentul la care s-a ajuns, un buton pentru redare conținut / oprire cu posibilitatea reluării sau nu precum și un control pentru reglarea volumului). În implementarea acestor cerințe funcționale, pot fi utilizate metodele `mediaPlayer.getStatus()`¹⁰⁹ care întoarce starea curentă a obiectului `MediaPlayer`, `mediaPlayer.seek()` care stabilește ca redarea conținutului să fie făcută începând de la momentul indicat ca parametru, `mediaPlayer.getStartTime()` care identifică momentul de început al conținutului multimedia, `mediaPlayer.setCycleCount()` care precizează numărul de redări.

Metodele de oportunitate pentru gestiunea evenimentelor ce pot apărea în procesul de redare al conținutului sunt `setOnPlaying`, `setOnPaused`, `setOnReady`, `setOnEndOfMedia`, toate primind ca argument un obiect de tip `Runnable`, reprezentând fire de execuție în care sunt realizate astfel de operații.

7. 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. Deși nu urmărește o schemă, FXML are o structură de bază predefinită. Astfel, cele mai multe clase JavaFX pot fi utilizate ca elemente și cele mai multe proprietăți (ale unor componente de tip JavaBeans) pot fi folosite ca atribute. Dintr-o perspectivă a arhitecturii MVC (Model View Controller), fișierul FXML care conține descrierea interfeței cu utilizatorul reprezintă partea de vizualizare. Controlul este realizat prin clasele Java¹¹⁰, 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 și pentru interfețe statice (formulare, controale, tabele). Pot fi definite și interfețe dinamice prin utilizarea de scripturi.

Printre beneficiile FXML se numără următoarele [10]:

- 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 și 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;

¹⁰⁹ Valorile pe care le poate întoarce această metodă sunt `Status.UNKNOWN`, `Status.HALTED`, `Status.PLAYING`, `Status.PAUSED`, `Status.READY` sau `Status.STOPPED`.

¹¹⁰ Aceasta poate implementa în mod opțional clasa `Initializable`, care este declarată drept control pentru fișierul FXML.

- conținutul unui fișier FXML poate fi localizat pe măsură ce este parcurs¹¹¹;
- 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 și o integrare cu limbaje pentru scripturi (ca JavaScript).

În versiunea 2.2 a JavaFX au fost introduse câteva facilități:

- eticheta `<fx:constant>` înlesnește procesul de căutare a constantelor, astfel încât constanta `java.lang.Double.NEGATIVE_INFINITY` definită în Java poate fi referită ca `<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 și sub-clasele care asigură controlul (în cadrul metodei `initialize`).
- inițializarea claselor care asigură controlul se face prin mecanismul de reflectare, instanța clasei `FXMLLoader` identificând automat metoda `initialize`¹¹² pe care o și apelează;
- ușurința în realizarea de noi controale FXML – prin intermediul metodelor `setRoot` și `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.*?>

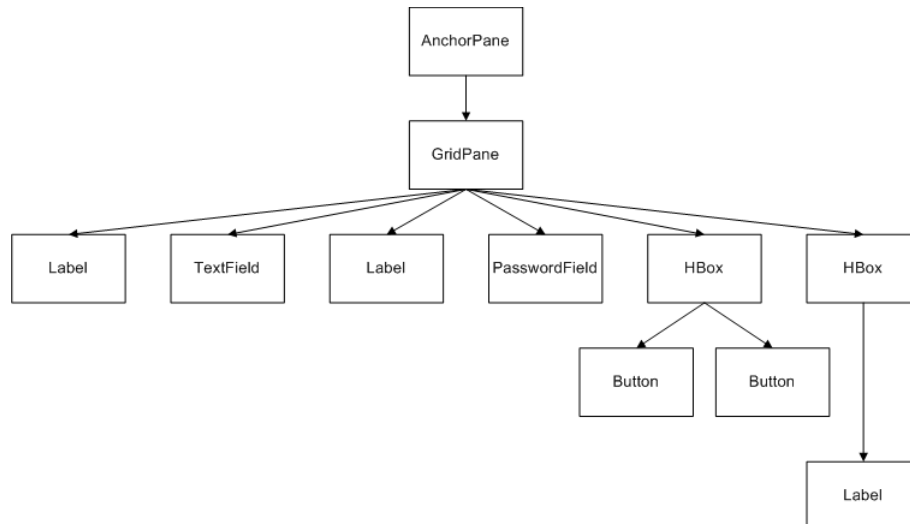
<AnchorPane fx:id="AnchorPane" prefHeight="150.0" prefWidth="400.0"
xmlns:fx="http://javafx.com/fxml" fx:controller="BookStore">
  <children>
    <GridPane layoutX="15.0" layoutY="15.0">
      <children>
        <Label text="Nume Utilizator:" GridPane.columnIndex="0"
GridPane.rowIndex="0" />
        <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">
```

¹¹¹ 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).

¹¹² În cazul în care nu este publică, pentru această metodă trebuie folosită adnotarea `@FXML`.

```
<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



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 și handleButonRenuntareAction. Acestea vor trebui adnotate cu șirul de caractere @FXML, ca și obiectele corespunzătoare nodurilor descrise în documentul respectiv.

Încărcarea documentului într-o scenă se va face astfel:

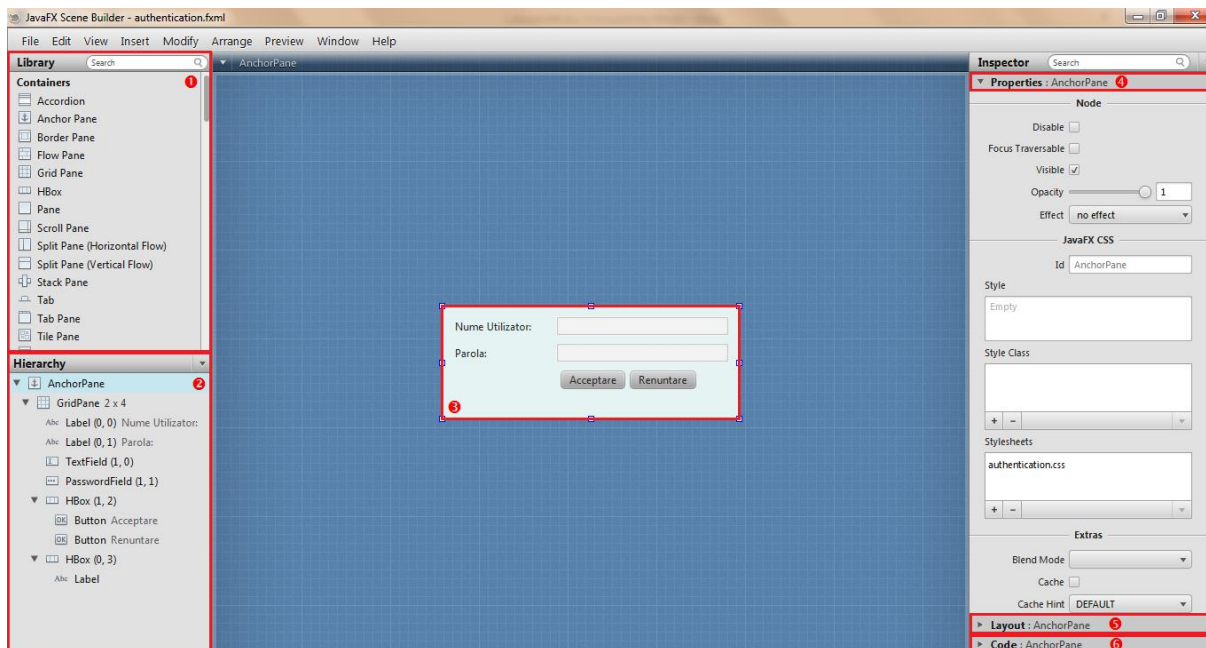
```
try {
    applicationScene =
        new Scene(
            (Parent) FXMLLoader.load(getClass().getResource("authentication.fxml"))
        );
} catch (Exception exception) {
    System.out.println("exception : "+exception.getMessage());
}
```

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

Documentele FXML pot fi dezvoltate în mod vizual folosind utilitarul SceneBuilder 1.1, disponibil pe platformele Windows (32/64 de biți), Mac OS X și Linux (32/64 biți) – pachete deb și arhive tar.gz. Acesta permite vizualizarea ierarhiei de noduri precum și stabilirea de diferite aspecte pentru fiecare nod în parte, grupate în proprietăți, moduri de dispunere și cod.

Caracteristicile utilitarului SceneBuilder sunt [11]:

- **interfață WYSIWYG de tip drag and drop** care permite crearea rapidă a unei interfețe cu utilizatorul fără a fi nevoie de a scrie cod sursă; controalele pot fi preluate dintr-o bibliotecă și dispuse în modul în care se dorește afișarea lor;
- **integrare facilă cu orice IDE pentru Java** de vreme ce este un utilitar de sine stătător;
- **generare automată a codului FXML** se produce la construirea interfeței cu utilizatorul, acesta fiind menținut separat de logica aplicației și foile de stil;
- **editare în timp real și funcționalități de previzualizare** ce permit ca modificările la interfața cu utilizatorul să fie vizibile pe măsură ce sunt implementate, fără a fi necesar ca aplicația să fie compilată, reducând timpul de dezvoltare al proiectului;
- **acces la întreaga bibliotecă de controale JavaFX 2.2;**
- **suport pentru CSS** permite gestiunea flexibilă a modului în care arată aplicația;
- **suport pentru mai multe sisteme de operare**, având pe toate un comportament consistent;
- **gratuitate** pentru toate tipurile de utilizatori.



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 și 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 construiește propriu-zis prin operații de tip drag and drop. Dacă nodul care se adaugă interfeței se află deasupra unui alt obiect¹¹³, 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).

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

În secțiunea 6 sunt indicate metodele executate din clasa controller atunci când o acțiune de tipul respectiv se produce asupra interfeței respective¹¹⁴.

9. Modele de date asociate controalelor JavaFX

JavaFX utilizează un model JavaBeans¹¹⁵ pentru reprezentarea obiectelor, oferind suport pentru proprietăți și pentru crearea de dependențe (*eng.* binding) între obiecte, exprimând relațiile dintre acestea¹¹⁶. O astfel de funcționalitate este extrem de utilă în dezvoltarea de interfețe grafice cu utilizatorul întrucât informațiile vizualizate sunt automat sincronizate cu sursa de date din care acestea sunt preluate [12].

O proprietate într-o anumită clasă care respectă acest model corespunde unui atribut al acesteia:

```
class MyClass {  
    private SomeTypeProperty myAttribute = new SomeTypeProperty();  
    // ...  
}
```

În situația în care clasa oferă suport pentru proprietăți, ea va implementa metodele setter și getter `setMyAttribute` respectiv `getMyAttribute` pentru stabilirea respectiv obținerea valorii propriu-zise a atributului respectiv, dar și metoda `myAttributeProperty` care va întoarce exact atributul în cauză. Pentru această valoare poate fi indicat un obiect de tip ascultător, ce monitorizează modificările realizate asupra acestuia.

¹¹³ 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.

¹¹⁴ Categoriile de evenimente incluse sunt drag and drop, evenimente legate de tastatură și mouse, precum și evenimente specifice dispozitivelor cu ecran tactil (derulare, rotație, atingere, modificarea rezoluției de vizualizare).

¹¹⁵ Modelul JavaBeans oferă suport pentru tipuri de proprietăți complexe și un sistem de transmitere al evenimentelor.

¹¹⁶ Atunci când un obiect participă într-o relație de tip dependență, modificările asupra sa sunt imediat vizibile la nivelul obiectului cu care se află în legătură. Interfețele de programare care oferă astfel de funcționalități pot fi de nivel înalt, în care dependențele se creează ușor și de nivel scăzut, folosite mai ales în cazul în care se dorește execuție rapidă și o utilizare redusă a memoriei.

Dependențele dintre obiecte pot fi realizate fie prin clase specifice tipului de date respectiv `DataTypeBinding` (unde `DataType` reprezintă tipul de date respectiv) fie direct prin clasa `Bindings`. De asemenea, sunt definite o serie de interfețe care permit obiectelor să fie notificate atunci când se modifică o valoare sau se produce invalidarea unor valori (`InvalidationListener`, `ChangeListener` primesc notificări produse de `Observable` și `ObservableValue`)¹¹⁷.

Implementarea proprietăților și a dependențelor în JavaFX suportă evaluarea leneșă (*eng.* lazy evaluation)¹¹⁸ ceea ce înseamnă că la momentul în care se produce o modificare, valoarea nu este recalculată imediat, ci numai dacă și atunci când valoarea în cauză este solicitată. Astfel, dependența este invalidă dacă s-au produs modificări și recalcularea nu s-a produs încă, fiind validată când acest proces este declanșat de solicitarea valorii.

```
MyClass myObject = new MyClass();
myObject.myAttributeProperty().addListener(new ChangeListener() {
    @Override
    public void changed(Observable Value oValue,
                       Object oldValue, Object newValue) {
        // ...
    }
});
```

În JavaFX sunt definite mai multe colecții de date în cadrul pachetului `javafx.collections` care conține [13]:

- **interfețe**

- `ObservableList` – o listă care permite ascultătorilor să identifice modificările la momentul la care se produc;
- `ListChangeListener` – o interfață care primește notificările modificărilor realizate asupra unui obiect de tip `ObservableList`;
- `ObservableMap` – o listă de asocieri cheie unică-valoare ce permite ascultătorilor să identifice modificările atunci când se produc;
- `MapChangeListener` – o interfață care primește notificările modificărilor realizate asupra unui obiect de tip `ObservableMap`.

- **clase**

- `FXCollections` – o clasă de utilitare ce pune la dispoziția programatorilor aceleași metode din clasa `java.util.Collections`;
- `ListChangeListener.Change` – reprezintă o schimbare realizată asupra unui obiect `ObservableList`;
- `MapChangeListener.Change` – reprezintă o schimbare realizată asupra unui obiect `ObservableMap`.

Interfețele `ObservableList` și `ObservableMap` sunt amândouă derivate din `javafx.beans.Observable` și `java.util.List`, respectiv `java.util.Map`, definind metode `addListener` primind parametri `ListChangeListener` / `MapChangeListener`.

¹¹⁷ Obiectul `ObservableValue` încapsulează și valoarea propriu-zisă și transmite modificările ei către orice obiect `ChangeListener` înregistrat în timp ce `Observable`, care nu conține și valoarea în sine transmite înregistrările către `InvalidationListener`.

¹¹⁸ Numai clasa `InvalidationListener` suportă atât evaluarea lentă cât și evaluarea rapidă, în timp ce clasa `ChangeListener` impune evaluarea rapidă (chiar dacă `ObservableValue` suportă și evaluarea leneșă) pentru că nu este posibil să se știe dacă valoarea a fost cu adevărat modificată până ce nu este recalculată.

Crearea unor astfel de obiecte se face prin metodele statice ale clasei `FXCollections.observableList` și `observableMap`, ce primesc ca argumente obiecte din clasele `java.util.List`, respectiv `java.util.Map`.

```
List<String> list = new ArrayList<String>();
ObservableList<String> observableList =
    FXCollections.observableList(list);
observableList.addListener(
    new ListChangeListener() {
        @Override
        public void onChanged
            (ListChangeListener.Change change) {
            while (change.next()) {
                if (change.wasAdded()) {
                    // ...
                }
                if (change.wasRemoved()) {
                    // ...
                }
                if (change.wasReplaced()) {
                    // ...
                }
                if (change.wasPermutated()) {
                    // ...
                }
            }
        }
    }
);
```

```
Map<String,String> map =
    new HashMap<String,String>();
ObservableMap<String,String> observableMap =
    FXCollections.observableMap(map);
observableMap.addListener(
    new MapChangeListener() {
        @Override
        public void onChanged
            (MapChangeListener.Change change) {
            // ...
        }
    }
);
```

De remarcat faptul că obiectul de tip `ListChangeListener.Change` poate conține informații despre mai multe modificări, motiv pentru care este necesar să se realizeze un iterator pe acestea prin apelarea metodei `next`, în timp ce `MapChangeListener.Change` conține o singură schimbare, corespunzătoare operației `put` sau `remove` care a fost realizată.

10. Gestiunea concurenței în JavaFX

Pachetul `javafx.concurrent` pune la dispoziție funcționalități pentru dezvoltarea de aplicații ce rulează pe mai multe fire de execuție, completând capabilitățile oferite de `java.util.concurrent` ținând cont de firul de execuție al aplicației JavaFX și de alte constrângeri specifice interfețelor grafice. În acest caz, ideea este de a oferi o viteză de răspuns satisfăcătoare prin execuția sarcinilor care consumă resurse către fire de execuție care rulează în fundal.

Graful de scene JavaFX nu poate fi accesat decât din firul de execuție responsabil de interfața cu utilizatorul (firul de execuție al aplicației JavaFX). Implementarea de sarcini care consumă resurse pe acest fir de execuție face ca interfața cu utilizatorul să fie nerespensivă motiv pentru care acestea trebuie alocate unuia sau mai multor fire de execuție care rulează în fundal, lăsând firul de execuție al aplicației JavaFX să se ocupe de evenimentele rezultate din interacțiunea cu utilizatorul.

Pachetul `javafx.concurrent` constă în interfața `Worker` și două clase de bază `Task` și `Service`, ambele implementând interfața `Worker` [14]. Aceasta oferă funcționalități necesare unui fir de execuție care rulează în fundal pentru a comunica cu interfața grafică. Clasa `Task`, implementare observabilă a clasei `java.util.concurrent.FutureTask`, permite specificarea de sarcini asincrone în timp ce clasa `Service` le execută. Clasa `WorkerStateEvent` generează evenimente de fiecare dată atunci când starea unui obiect `Worker` se modifică.

Clasele `Task` și `Service` implementează interfața `EventTarget` astfel încât suportă ascultarea evenimentelor legate de starea acestor fire de execuție.

Interfața `Worker` definește un obiect care realizează anumite sarcini pe unul sau mai multe fire de execuție, starea sa fiind observabilă din contextul firului de execuție al aplicației JavaFX. Stările pe care le poate avea un obiect care implementează această interfață sunt `READY` (imediat după ce a fost creat), `SCHEDULED` (atunci când este programat pentru o anumită sarcină), `RUNNING` (după ce a fost pornit)¹¹⁹, având, la terminarea sarcinii, în funcție de rezultatul obținut, una din stările `SUCCEEDED` (dacă sarcina este executată cu succes, proprietatea `value` fiind transmisă rezultatului acestui obiect), `FAILED` (dacă sarcina nu a putut fi finalizată, proprietatea `exception` fiind transmisă excepției care s-a produs) sau `CANCELED` dacă sarcina este întreruptă de utilizator prin apelarea metodei `cancel`. Progresul realizat pe parcursul execuției se poate obține prin diferite proprietăți precum `totalWork`, `workDone` și `progress`.

Clasa `Task` este folosită pentru a implementa logica sarcinii care trebuie executată pe un fir de execuție în fundal. Implementările clasei `Task` trebuie să suprascrîie metoda `call` pentru a realiza operațiile dorite și pentru a furniza rezultatul. Fiind invocată pe un fir de execuție în fundal, aceasta nu poate accesa decât proprietăți publice din cadrul firului de execuție al aplicației JavaFX, prin intermediul metodelor `updateProgress`, `updateMessage` și `updateTitle`, în caz contrar generându-se excepții. În situația în care sarcina este întreruptă, rezultatul metodei `call` e ignorat. Derivată din `java.util.concurrent.FutureTask` care implementează interfața `Runnable`, clasa `Task` permite ca obiectele sale să fie transmise unui fir de execuție ca parametru sau să fie folosite din interfața de programare `Executor`. De asemenea, obiectul de tip `Task` poate fi apelat prin metoda `FutureTask.run` din cadrul altui fir de execuție ce rulează în fundal.

```
Thread thread = new Thread(task);  
thread.setDaemon(true);  
thread.start();  
  
ExecutorService.submit(task);
```

Un obiect de tip `Task` trebuie să poată fi oprit de fiecare dată când se apelează metoda `cancel` din contextul său, de aceea, în metoda `call`, acesta trebuie să verifice periodic dacă nu cumva s-a produs un astfel de eveniment, folosind metoda `isCanceled()`. În cazul în care în metoda `call` există apeluri blocante de tipul `Thread.sleep`, și metoda `cancel` este apelată în timpul execuției acestei metode, se va genera o excepție `InterruptedException` astfel încât tratarea excepției trebuie să verifice și motivul care a generat-o.

Vizualizarea progresului unui obiect de tip `Task` în cadrul unui control aparținând firului de execuție al aplicației JavaFX poate fi realizată prin operația de asociere (`bind`) între acestea, pe baza proprietății de progres, actualizată în cadrul metodei `call` prin metoda `updateProgress`¹²⁰.

Întrucât clasa `Task` definește un obiect care nu poate fi reutilizat, pentru implementarea unei astfel de funcționalități trebuie folosită clasa `Service`.

¹¹⁹ Și în cazul în care firul de execuție este pornit în mod direct, el va trece mai înainte prin starea `SCHEDULED` înainte de `RUNNING`, chiar dacă nu a fost programat propriu-zis.

¹²⁰ Metoda `updateProgress` actualizează mai multe proprietăți, cum ar fi `progress`, `totalWork` sau `workDone` astfel încât poate fi apelată metoda `progressProperty` pentru a obține progresul sarcinii.

Aceasta este proiectată să execute un obiect de tip `Task` pe unul sau mai multe fire de execuție care rulează în fundal. Scopul acestei clase este de a ajuta programatorul să implementeze interacțiunea corectă dintre firele de execuție din fundal și firul de execuție al aplicației JavaFX astfel încât metodele și stările unei astfel de clase nu pot fi accesate decât din acest context. Un obiect `Service` poate fi pornit (prin metoda `Service.start()`), anulat sau repornit, observându-se, pe parcursul rulării sale starea proceselor din fundal. Atunci când se implementează subclase `Service`, parametrii de intrare trebuie expuși către obiectul de tip `Task` ca proprietăți ale acesteia. Un obiect de tip `Service` poate fi executat printr-un obiect `Executor`, printr-un fir de execuție de tip `daemon` sau printr-un anumit tip de executor cum ar fi `ThreadPoolExecutor`.

De fiecare dată când starea unui obiect de tip `Worker` se schimbă, se produce un eveniment corespunzător, definit de clasa `WorkerStateEvent`: `ANY`, `WORKER_STATE_CANCELLED`, `WORKER_STATE_FAILED`, `WORKER_STATE_READY`, `WORKER_STATE_RUNNING`, `WORKER_STATE_SCHEDULED`, `WORKER_STATE_SUCCEEDED`. Acestea apelează metodele de tratare aferente și apoi mecanismele de oportunitate `cancelled`, `failed`, `ready`, `running`, `scheduled`, `succeeded`, acestea fiind invocate pentru tranzițiile în stările respective. Metodele pot fi suprascrise de subclasele derivate din `Task` și `Service` pentru a implementa logica aplicației conform cerințelor funcționale.

11. Câteva recomandări cu privire la implementarea aplicațiilor JavaFX

Recomandările referitoare la implementarea aplicațiilor JavaFX vizează atât nivelul de logică a aplicației, cât și nivelul de prezentare.

În privința nivelului de logică a aplicației [15],

- atunci când trebuie încărcate mai multe resurse într-o scenă, operația implicând o perioadă de timp considerabilă, trebuie folosite ecrane animate care să indice progresul acestui proces astfel încât aplicația să nu pară că nu răspunde la interacțiunea cu utilizatorul;
- folosirea unor denumiri simbolice pentru pachetele implementate pentru ca aplicația să fie mai organizată și mai ușor de întreținut;
- implementarea arhitecturii model-view-controller (MVC) prin intermediul fișierelor FXML pentru definirea scenelor, acestea reprezentând partea de vizualizare, în timp ce modelul este reprezentat de obiectele de domeniu specifice aplicației, iar partea de control aparține codului Java care definește comportamentul interfeței grafice în interacțiunea cu utilizatorul
- rularea sarcinilor care implică procesarea unui volum mare de date pe fire de execuție separate, care rulează în fundal, pentru a evita ca utilizatorul să aștepte foarte mult până la terminarea metodei.

Referitor la nivelul de prezentare [16],

- utilizarea panourilor de dispunere, astfel încât conținutul să fie organizat și să poată fi utilizat cu ușurință;
- asigurarea faptului că toate butoanele au aceeași dimensiune se poate realiza mai ușor prin încadrarea tuturor într-un panou de dispunere de tip `VBox` sau prin indicarea valorilor maxime pentru dimensiunile lor și dispunerea într-un obiect de tip `TilePane`;

- menținerea nodurilor la dimensiunile preferate (`Control.USE_PREF_SIZE`) astfel încât în cazul unor operații de redimensionare să nu se producă comportamente nedorite;
- prevenirea operației de redimensionare prin specificarea dimensiunilor minime, maxime și preferate la aceeași valoare;
- alinierea nodurilor și panourilor folosind metoda `setAlignment` care primește ca parametru constante din pachetul `javafx.geometry` precum `HPos` pentru alinierea orizontală, `VPos` pentru alinierea verticală sau `Pos` pentru alinierea verticală și orizontală¹²¹;
- folosirea foilor de stil pentru definirea aspectului interfeței grafice, aceasta putând fi modificată cu ușurință prin schimbarea foi de stil asociate.



Activitate de Laborator

Pentru rezolvarea acestui laborator, trebuie să aveți instalate:

- Java 7, update 45 (JavaFX 2.2.45)
- NetBeans 7.4 / Eclipse 4.3.1¹²² (Kepler)
- Scene Builder 1.1

Se dorește 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, ștergere, căutare).

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

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 fișierul `authentication.fxml` cu SceneBuilder 1.1 sau cu orice alt editor de text – inclusiv în cadrul NetBeans / Eclipse.

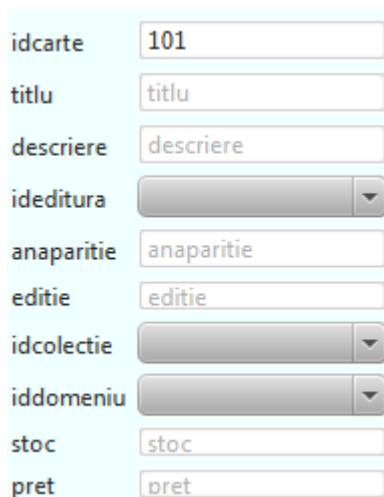
¹²¹ Această constantă are forma `Pos.VALIGN_HALIGN` unde `VALIGN` poate fi `TOP`, `BOTTOM`, `CENTER`, iar `HALIGN` poate lua valorile `LEFT`, `RIGHT`, `CENTER`.

¹²² Pentru proiectul Eclipse, se recomandă ca biblioteca `jfxrt.jar` să fie preluată de la `%SystemRoot%\Program Files\Java\jdk1.7.0_45\jre\lib` (sistemele Windows), respectiv de la `$JAVA_HOME\jdk1.7.0_45\jre\lib` (sistemele Linux).

- a. Pentru nodul rădăcină de tip `AnchorPane`, să se indice drept controller clasa `BookStore`.
- b. Pentru butoanele `Acceptare` / `Renunțare`, să se specifice metodele care vor fi executate în cazul producerii unui eveniment de tipul `ActionEvent.ACTION`.
- c. În clasa `BookStore`, să 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 afișa un mesaj corespunzător în `etichetaAfisare`. Obiectele și metodele definite în documentul FXML pot fi accesate din clasa controller dacă sunt adnotate cu șirul 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`).



The image shows a form for registering a book. It consists of a list of labels on the left and corresponding input fields on the right. The labels are: idcarte, titlu, descriere, ideditura, anaparitie, editie, idcolectie, iddomeniu, stoc, and pret. The input fields are: a text box containing '101' for idcarte, a text box containing 'titlu' for titlu, a text box containing 'descriere' for descriere, a dropdown menu for ideditura, a text box containing 'anaparitie' for anaparitie, a text box containing 'editie' for editie, a dropdown menu for idcolectie, a dropdown menu for iddomeniu, a text box containing 'stoc' for stoc, and a text box containing 'pret' for pret.

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, ștergere) 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 primare¹²³.

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

¹²³ Această regulă se aplică doar în cazul cheilor primare autoincrementale, deci nu și pentru tabela utilizatori.

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

[1p] 7. Să se creeze o animație asupra butonului deasupra căruia se găsește mouse-ul astfel încât acesta să aibă un efect de tip umbră exterioară (`DropShadow`) implementând și o tranziție care pe parcursul unei secunde modifică transparența de la 1.0 la 0.8 și scalează dimensiunile butonului cu 0.1. Animația trebuie să ruleze tot timpul cât mouse-ul se găsește deasupra butonului având totodată un caracter ciclic.

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] Creating Transitions and Timeline Animations in JavaFX, <http://docs.oracle.com/javafx/2/animations/jfxpub-animations.htm>
- [7] Using JavaFX Charts, <http://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm>
- [8] Adding HTML Content to JavaFX Applications, <http://docs.oracle.com/javafx/2/webview/jfxpub-webview.htm>
- [9] Incorporating Media Assets into JavaFX Applications, <http://docs.oracle.com/javafx/2/media/jfxpub-media.htm>
- [10] Mastering FXML, http://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm
- [11] Overview of JavaFX SceneBuilder, <http://docs.oracle.com/javafx/scenbuilder/1/overview/jsbpub-overview.htm>
- [12] Use JavaFX Properties and Binding, <http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm>
- [13] Using JavaFX Collections, <http://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm>
- [14] Concurrency in JavaFX, <http://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm>
- [15] Implementing JavaFX Best Practices, http://docs.oracle.com/javafx/2/best_practices/jfxpub-best_practices.htm
- [16] Tips for Sizing and Aligning Nodes, http://docs.oracle.com/javafx/2/layout/size_align.htm