**UNIVERSITATEA DE STAT DIN MOLDOVA**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**DEPARTAMENTUL ”INFORMATICA”**

**Ciolpan Vlad**

**Prelucrarea evenimentelor în Java. Event oriented programing.**

**IA2001**

**Teză de an**

Chișinău-2022

**Cuprins**

**Lista abrevierilor**

**Introducere**

**I. MATERIAL TEORETIC:**

* 1. Descriere
  2. Definirea evenimentului
  3. Componente grafice
  4. Tipuri de evenimente și cum se generează
  5. Prelucrarea unui eveniment

**II. DESCRIEREA INSTRUMENTELOR SOFT UTILIZATE:**

* 1. IDE pentru scrierea codului
  2. Scene Builder

**III. DESCRIEREA PĂRȚII PRACTICE:**

**Concluzii**

**Bibliografie**

**Anexe**

**Lista abrevierilor**

AWT – Abstract Window Toolkit

IDE – Integrated Development Environment

JAR – Java ARchive

MVC - Model–View–Controller

**Introducere**

**I MATERIAL TEORETIC**

**1.1 Descriere**

Programarea orientată pe eveniment a apărut ca cosecință la necesitatea de a crea ușor și comod aplicații care cer utilizatorul să intercționeze cu o interfață grafică. Astfel în loc de citirea doar a datelor de intrare al calculatorului, se generează evenimente ce conțin informații utile despre sine. Generarea evenimentelor presupune și prelucrearea lor. Astfel în momentul prelucrării, se pot descrie acțiunile efectuate de către calculator.

Datorită acestei abordări devine posibil menținerea codului organizat. Un alt plus al acestei tehnologii împreună cu structurizarea componentelor grafice în formă de arbore conduce la performanță rapidă la interacțiunea utilizatorului cu aplicația.

În continuare va fi descrisă pe etape ce este un eveniment, ce componente grafice există pentru a oferi context tipurilor de evenimente, generării și prelucrării lor.

**1.2 Definirea evenimentului**

Eveniment – rezultatul interacțiunii utilizatorului cu aplicația ce are interfață grafică. De exemplu la apăsarea unui buton de pe tastatură, la mișcarea cursorului pe ecran.

În limbajul Java la baza oricărui eveniment se află clasa java.util.EventObject, precum și toate clasele din Java moștenesc de la clasa de bază java.lang.Object. Această clasă conține un singur câmp:

1. „*source*” de tip java.lang.Object – ce păstrează informația despre obiectul care a chemat acest eveniment.

și o metodă:

1. „*getSource()*” – returnează informația din câmpul „*source*”.

În continuare există 2 încrengături: java.awt.AWTEvent și javafx.event.Event, care moștenesc de la java.util.EventObject. Java.awt.AWTEvent este clasa de bază pentru toate evenimentele din instrumentarul AWT. Iar javafx.event.Event este clasa de bază pentru toate evenimentele folosite în JavaFX. În această lucrare s-a folosit JavaFX, de aceea în continuare se vor descrie evenimentele și modul lor de lucru doar pentru această tehnologie.

Precum a fost menționat javafx.event.Event este clasa de la care moștenesc evenimentele folosite în JavaFX. Astfel din punctul de vedere al computer-ului eveniment este un obiectul de tipul acestei clase sau al altor subclase care moștenesc de la această clasă.

Clasa javafx.event.Event conține câmpuri ce păstrează informația de bază despre eveniment necesară pentru funcționarea și la prelucrarea sa:

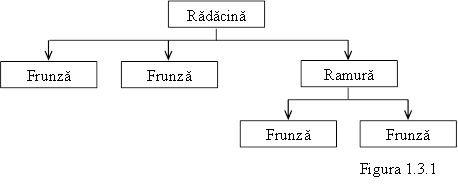
1. „*eventType*” de tipul EventType<? extends Event> – păstreză informația despre tipul evenimentului
2. „target” de tipul EventTarget – joacă rol în construirea căii de parcurgere a evenimentului
3. „*consumed*” de tipul boolean – dacă evenimentul a fost prelucrat sau nu

și metodele respective acestor câmpuri:

1. „*getEventType()*” – returnează informația din câmpul „*eventType*”;
2. „*getTarget()*” – returnează informația din câmput „target”;
3. „*isConsumed()*” – returnează informația din câmput „*consumed*”;
4. „*consume()*” – setează câmpul „*consumed*” cu valoarea true, ceea ce indică că evenimentul a fost prelucrat;
5. „*fireEvent(EventTarget eventTarget, Event event)*” – metoda statică care permite transmiterea unui eveniment „*event*” pe o anumită cale „*eventTarget*”.

**1.3 Componente grafice**

Componentă grafică se referă la toate componentele vizuale din care este formată o aplicație. În JavaFx toate componentele se exprimă ca noduri organizate în formă de arbore. Nodurile se împart în 3 tipuri: rădăcină, ramură și frunză. Nodul ramură este nodul care poate avea copii. Nodul rădăcină este un nod ramură, dar care are părinte. Nodul frunză este nodul care nu poate avea copii.



Clasa de bază pentru orice nod reprezintă clasa abstractă javafx.scene.Node care moștenește de la clasa java.lang.Object. Această clasă implementează interfața javafx.event.EventTarget care a fost menționată în primul paragraf în cadrul clasei javafx.event.Event necesar pentru construcția căii parcurgerii evenimentelor.

Clasa javafx.scene.Node are o mulțime de proprietăți care se rezumă la: id-ul, proprietățile fizice (locația și translația în sistemul de coordonate, mărimea, rotația), ascultărorii de evenimente, vizibilitate ș.a. Respectiv cu fiecare proprietate există și metodele care oferă acces la aceste proprietăți.

Clasa de bază pentru orice nod părinte reprezintă clasa abstractă javafx.scene.Parent care moștenește de la clasa javafx.scene.Node. Această clasă conține toate metodele necesare pentru a manipula cu ierahia de noduri, cum ar fi adaugărea și scoaterea nodurilor copii.

Nodurile frunze sunt caracterizate de orice altă clasă derivată de la clasa javafx.scene.Node înafară de clasa javafx.scene.Parent, cum ar fi:

1. javafx.scene.shape.Shape – clasa de bază pentru orice figură geometrică disponibilă
2. javafx.scene.shape.Shape3D – clasa de bază pentru orice corp geometric disponibil
3. javafx.scene.canvas.Canvas – clasă care permite „desenul” pe câmpul acestui nod
4. javafx.scene.image.ImageView – clasă care permite afișarea imaginilor
5. javafx.scene.media.MediaView – clasă care permite afișarea unui video sau pentru audio

De la clasa javafx.scene.Parent se extind 2 clase: javafx.scene.Group și javafx.scene.layout.Region. Group acționează ca un tot întreg, adică stilizând un Group se vor stiliza și nodurile copii al acestui nod. Pe când Region permite stilizarea fiecărui nod copil aparte.

Clasa javafx.scene.layout.Region reprezintă clasa de bază pentru componentele ce oferă control utilizatorului, cum ar fi buton, și pentru containerele care reprezintă componente ce pot păstra în interiorul lor alte componente. Ca proprietăți această clasă conține informarțiile necesare pentru stilizarea și afișarea unei componente, cum ar fi lungimea, lățimea, marginea ș.a. Respectiv fiecărei proprietăți corespund metodele necesare pentru manipularea informației din interior.

De la clasa javafx.scene.layout.Region moștenesc 4 clase:

1. javafx.scene.chart.Axis<T> – clasă de bază pentru axa de coordonate Oxy
2. javafx.scene.chart.Chart – clasă de bază pentru diagrame
3. javafx.scene.control.Control – clasă de bază pentru controale
4. javafx.scene.layout.Pane – clasă care reprezintă cel mai de bază container

Clasa javafx.scene.layout.Pane moștenește toate proprietățile de la clasa javafx.scene.layout.Region, astfel având proprietățile și metodele minime necesare pentru a putea funcționa ca un container.

De la această clasă moștenesc toate containerele existente în JavaFX, fiecare cu o funcțiune specială:

1. javafx.scene.layout.AnchorPane – panou în care componentele copii sunt fixate de panou la o anumită distanță față de margini, astfel permițând aranjarea componentelor mai ușoară
2. javafx.scene.layout.BorderPane – panou împărțit în zone, sus, jos, stânga, dreapta, centru, astfel în fiecare zonă pot fi amplasate câte o componentă sau să rămână zonă liberă
3. javafx.scene.control.DialogPane – panou considerat ca nod rădăcină, care apare ca fereastră aparte de aplicație, des folosit pentru a confirma o acțiune
4. javafx.scene.layout.FlowPane – panou în care componentele copii se aranjează într-un flux fiind separate orizontal și vertical de o valoare specificată
5. javafx.scene.layout.GridPane – panou împărțit în rânduri și coloane, formând celule neuniforme în care sunt amplasate componenetele copii
6. javafx.scene.layout.HBox – panou în care componentele copii se aranjează orizontal unul după altul după un anumit spațiu specificat
7. javafx.scene.layout.VBox – panou în care componentele copii se aranjează vertical unul după altul după un anumit spațiu specificat
8. javafx.scene.layout.StackPane – panou în care componentele copii se vizualizează ordonate în formă de stivă, astfel ultima componentă adăugată este prima care se vede
9. javafx.scene.text.TextFlow – panou în care componentele textuale se aranjează într-un flux,
10. javafx.scene.layout.TilePane – panou împărțit în rânduri și coloane, formând celule uniforme în care sunt amplasate componenetele copii

Clasa abstractă javafx.scene.control.Control este clasa de bază pentru toate componentele cu care poate interacționa un utilizator. Aceasta conține 3 câmpuri noi:

1. „*contextMenu*” – meniu care apare la apasărarea al 2 buton de la șoricel
2. „*skin*” – pentru a stiliza înfățișarea controalelor
3. „*tooltip*” – pentru a afișa o mică informație când cursorul trece deasupra componentei

și metodele respective pentru a seta și citi informația din aceste câmpuri.

De la această clasă moștenesc următoarele clase:

1. javafx.scene.control.Labeled – clasă abstractă de bază pentru toate componentele asociate cu un text care nu poate fi editat. Conține câmpuri și metode necesarea pentru formatarea textului
2. javafx.scene.control.ChoiceBox<T> – boxă care oferă utilizatorului să aleagă un element dintr-o listă mică cu valori predefinite
3. javafx.scene.control.ComboBoxBase<T> – clasă abstractă de bază pentru componente ce oferă o alegere ca ChoiceBox, dar în același timp și posibilitatea de a introduce date
4. javafx.scene.web.HTMLEditor – zonă de introducere a caracterelor de pe tastarură și formatarea lor
5. javafx.scene.control.ListView<T> – listă de elemente ce pot fi aranjate vertical sau orizontal, din care utilizatorul poate alege
6. javafx.scene.control.Pagination – permite navigarea prin pagini, unde ca pagină se referă la un nod și index-ul corespunzător. Paginile se crează cu ajotorul la „*Page factory*”
7. javafx.scene.control.ProgressIndicator – indică un progres în formă circulară, având valori reale de la 0 la 1. Subclasa javafx.scene.control.ProgressBar oferă aceeași informație, doar sub formă de bară
8. javafx.scene.control.ScrollBar – bară de defilare, orizontală sau verticală, des folosit în sisteme mai complexe, cum ar fi ScrollPane sau ListView.
9. javafx.scene.control.ScrollPane – panou cu bări de defilare
10. javafx.scene.control.Separator – linie,orizontală sau verticală, ce separă conținutul
11. javafx.scene.control.Slider – oferă posibilitatea de a introduce o valoare numerică folosind o axă de la valoare minimă până la cea maximă
12. javafx.scene.control.Spinner<T> – câmp textul care permite alegerea unei valori dintr-o secvență de valori
13. javafx.scene.control.SplitPane – panou împărțin de unul sau mai mulți separatori
14. javafx.scene.control.TableView<S> – tabel format din rânduri, coloane, celule
15. javafx.scene.control.TabPane – panou ce permite deplasarea între file
16. javafx.scene.control.TextInputControl – clasă abstractă ce conține datele necesarea pentru a crea baza pentru câmpuri ce permit introducerea de text de la tastatură
17. javafx.scene.control.ToolBar – bară, orizontală sau verticală, pe care pot fi amplasate orice nod, cel mai des Button, ToggleButton și Separator
18. javafx.scene.control.MenuBar – bară formată din javafx.scene.control.Menu-uri, de obicei aranjată la cea mai de sus poziție în aplicație
19. javafx.scene.control.TreeView<T> – afișare de informație sub formă unui arbore
20. javafx.scene.control.TreeTableView<S> a – combinație dintre TreeView și TableView

De la clasa javafx.scene.control.Labeled moștenesc următoarele clase:

1. javafx.scene.control.Label – zonă cu text care nu poate fi editat
2. javafx.scene.control.ButtonBase – clasă abstractă de bază pentru toate butoanele
3. javafx.scene.control.TitledPane – panou cu titlu care poate fi deschis/închis. Poate fi inclus într-un grup, javafx.scene.control.Accordion, de TitledPane-uri, în care doar un singur panou poate fi afișat simultan

De la clasa javafx.scene.control.ButtonBase moștenesc următoarele clase:

* + 1. javafx.scene.control.Button – reprezintă un simplu buton care poate fi apăsat. Conține 2 câmpuri de tip boolean care indică alte 2 moduri de interacțiune cu butonul: „*cancelButton*” și „*defaultButton*”. Butonul cu „*cancelButton*” setat pe true va reacționa la apăsararea butonului ESC de pe claviatură, cu condiția că nici un alt component nu reacționeză la această apăsare. La fel și butonul cu „*defaultButton*” setat pe true va reaționa la apăsarea butonului ENTER. Poate fi inclus într-un grup, javafx.scene.control.ButtonBar, unde toate butoanele se aranjează ca într-un HBox
    2. javafx.scene.control.CheckBox –casetă pentru bifat și text corenspunzător. Conține 2 câmpuri de tip boolean: „*indeterminate*” și „*selected*” care generează una din cele 3 stări: selectat(false, true), neselectat(false,false) și nedefinit(true, –)
    3. javafx.scene.control.Hyperlink – adresă către o pagină Web. Conține un singur câmp de tip boolean „*visited*” ce înseamnă a fost vizitat adresa sau nu
    4. javafx.scene.control.MenuButton – buton care cheamă un „*contextMenu*”
    5. javafx.scene.control.ToggleButton – buton care poate fi selectat. La apăsarea repetată butonul devinde neselectat. Poate fi inclus în grupă, „*ToggleGroup*”, cu alte ToggleButton-uri în care doar un singur buton poate fi selectat. Subclasa javafx.scene.control.RadioButton are specific înfățișarea și faptul că la repetată pe un buton selectat acesta nu devine neselectat.

De la clasa javafx.scene.control.ComboBoxBase<T> moștenesc următoarele clase:

1. javafx.scene.control.ComboBox<T> – boxă care oferă utilizatorului să aleagă un element dintr-o listă mai mare cu valori predefinite. Este permis căutarea valorilor din listă prin intermediul câmpului predispus. Lista poate avea mai multe valori datorită barei de defilare.
2. javafx.scene.control.ColorPicker – permite alegerea unei culori
3. javafx.scene.control.DatePicker – permite alegerea unei date

De la clasa javafx.scene.control.TextInputControl moștenesc următoarele clase:

1. javafx.scene.control.TextArea – câmp ce permite introducerea de text pe mai multe rânduri
2. javafx.scene.control.TextField – câmp ce permite introducerea de text pe un singur rând. Subclasa javafx.scene.control.PasswordField ascunde textul introdus

Clasa javafx.scene.control.Menu menționată anterior este folosită pentru a forma meniuri din javafx.scene.control.MenuItem-uri sau clase care derivă de la ea. La cerința utilizatorului prin apăsarea pe obiect de tip Menu se deschide lista de MenuItem-uri la interacțiunea cu care meniul se închide. Astfel utilizatorul poate alege doar un singur element din meniu pe odată.

**1.4 Tipuri de evenimente și cum se generează**

Precum am stabilit în primul paragraf, eveniment este rezultat al interacțiunii utilizatorului cu aplicația. Pentru fiecare tip de interacțiune există un anumit tip de evenimente care derivă de la clasa de bază javafx.event.Event. În continuare se va enumăra lista de evenimente:

1. javafx.scene.input.InputEvent – clasă de bază pentru evenimentele ce reprezintă date de intrare de la utilizator. Toate clasele copii ai aceste clase sunt generate și prelucrate de orice subclasă a clasei javafx.scene.Node, adică orice componenetă al aplicației.
2. javafx.event.ActionEvent – eveniment generat în urma interacțiunii cu butoane. Este generat de subclasele claselor ButtonBase și MenuItem – când se apasă butonul, sublcasele clasei ComboBoxBase<T> – când se schimbă elementul selectat, clasa TextField – când este apăsată tasta ENTER
3. javafx.scene.control.DialogEvent – eveniment generat la interacțiunea cu fereastra de dialog, clasa Dialog<R> ce reprezintă o conexiunea dintre DialogPane – partea grafică a ferestrei de dialog și partea logică a programului
4. javafx.scene.media.MediaErrorEvent – eveniment generat la apariția unei erori cu manipularea resurselor audio sau video în clasa MediaView
5. javafx.scene.control.ScrollToEvent<T> – eveniment generat de toate componentele ce au un ScrollBar cum ar fi ListView, TableView, TreeView și TreeTableView la cerința utilizatorului de a derula până va fi văzut elementul necesar
6. javafx.scene.control.SortEvent<C> – eveniment generat de TableView și TreeTableView la chemarea funcției „*sort()*”
7. javafx.stage.WindowEvent – eveniment generat la interacțiunea cu fereastra aplicației
8. javafx.scene.transform.TransformChangedEvent – eveniment generat de orice Node la schimbarea propriităților fizice a componentului, cum ar fi mărimea, rotația, translația
9. javafx.scene.control.ListView.EditEvent<T> – eveniment generat la începutul, amânarea și săvârșirea editării celulelor dintr-un ListView
10. javafx.scene.control.TableColumn.CellEditEvent<S,T> – eveniment generat la începutul, amânarea și săvârșirea editării celulelor dintr-un TableView
11. javafx.scene.control.TreeView.EditEvent<T> – eveniment generat la începutul, amânarea și săvârșirea editării celulelor dintr-un TreeView
12. javafx.scene.control.TreeItem.TreeModificationEvent<T> – eveniment generat la modificarea strării elementelor dintr-un TreeView
13. javafx.scene.control.CheckBoxTreeItem.TreeModificationEvent<T> – eveniment generat în TreeView cu elementele arborelui în formă de CheckBox la schimbararea valorilor selecetate
14. javafx.scene.control.TreeTableView.EditEvent<S> - eveniment generat la începutul, amânarea și săvârșirea editării asupra la un TreeTableView
15. javafx.scene.control.TreeTableColumn.CellEditEvent<S,T> – eveniment generat la începutul, amânarea și săvârșirea editării celulelor dintr-un TreeTableView

Clasele care moștenesc de la clasa javafx.scene.input.InputEvent:

1. javafx.scene.input.ContextMenuEvent – eveniment generat la cererea utilizatorului de a afișa „*ContextMenu*” al nodului. Conține coordonatele acestui menu pe ecran și relativ de sursa evenimentului
2. javafx.scene.input.DragEvent – eveniment generat la tragerea și plasarea datelor în sau din aplicație. Conține datele necesare pentru detectarea intrării în, stării deasupra sau ieșirii din Node-uri
3. javafx.scene.input.InputMethodEvent – eveniment generat de modificarea textului din Node-urile care moștenesc de la clasa TextInputControl
4. javafx.scene.input.KeyEvent – eveniment generat la apăsarea butoanelor de pe tastatură. Conține date despre caracterul, codul butonului apăsat și dacă sunt apăsate tastele SHIFT, CTRL, ALT
5. javafx.scene.input.MouseEvent – eveniment generat la manipularea cu ajutorul mouse-ului și cursorului. Conține date despre coordonatele sursei care a generat evenimentul. Include apăsarea și eliberării butonului mouse-ului, intrării și ieșirii cursorului prin Node-uri, detectării tragerii de Node-uri, adică mișcarea mouse-ului cu butonul apăsat deasupra unui Node. Dacă este necesară implicarea mai multor Node-uri în mișcarea de tragere și plasare se generează subclasa javafx.scene.input.MouseDragEvent ce conține aceleași informații ca DragEvent
6. javafx.scene.input.GestureEvent – eveniment generat la manipularea utilizatorului cu tableta grafică. Conține coordonatele sursei care a generat evenimenutul. Subclasele acestei clase reprezintă manipulări specifice cum ar fi RotateEvent – generat când utilizatorul încearcă cu 2 degete să rotească Node-ul, ScrollEvent – generat când utilizatorul încearcă să ruleze conținutul, SwipeEvent – generat când utilizatorul mișcă cu degetul într-o direcție , ZoomEvent – generat când utilizatorul încearcă să mărească conținutul prin mișcarea a 2 degete de la un punct comun în părți diferite.
7. javafx.scene.input.TouchEvent – eveniment generat la apăsarea utilizatorului pe tableta grafică. Conține date despre punctul unde a fost apăsat, modul în care a fost efectuată apăsarea – apăsare simplă, apăsarea a fost eliberată, apăsare și ținere.

**1.5 Prelucarea unui eveniment**

În procesul de prelucrare a unui eveniment se urmăresc următorii pași:

1. Generarea unui eveniment
2. Selectarea țintei
3. Construncția rutei
4. Capturarea evenimentului
5. „Clocotirea” evenimentului
6. Consumarea evenimentului

Generarea unui eveniment a fost descris în paragraful precendent pentru fiecare tip de eveniment.

Pentru selectarea țintei este necesar de interfața EventTarget ce conține metoda „*buildEventDispatchChain()*” folosită la construcția rutei. Orice clasă începând cu clasa de bază Node implementează această interfață.

Sunt anumite reguli, în depedență de tipul evenimentului, după care se alege Node-ul țintă:

1. Dacă este KeyEvent atunci se alege ca țintă Node-ul luat în focalizare
2. Dacă este MouseEvent atunci se alege ca țintă Node-ul sub cursor
3. Dacă este GestureEvent citit de un ecran tactil cu mișcări continuu atunci se alege ca țintă Node-ul aflat în centrul tuturor atingerilor de la începutul mișcării
4. Dacă este GestureEvent citit cu ajutorul la o tabletă grafică atunci se alege ca țintă Node-ul aflat sub cursor
5. Dacă este SwipeEvent citit de un ecran tactil atunci se alege ca țintă Node-ul aflat în centrul traseului parcurs de către degete în momentul efectuării glisării
6. Dacă este TouchEvent atunci se alege ca țintă Node-ul aflat în locul unde a fost efectuată prima atingere
7. Dacă sunt mai multe Node-uri suprapuse unul peste altul, atunci se alege ca țintă Node-ul aflat deasupra tutor.

Construcția rutei de parcurgere a evenimentului are loc cu ajutorul metodei „*buildEventDispatchChain()*” care este implementată de orice Node datorită interfaței EventTarget. Această rută pornește de la Node-ul de bază până nu ajunge la Node-ul țintă.

La etapa de capturare a evenimentului, acesta parcurge ruta construită de la Node-ul de bază până la Node-ul țintă. La această etapă evenimentul este prelucrat de către filtrele Node-urilor alfate pe această rută, dacă acestea au fost adaugate.

La etapa de „clocotire” a evenimentului, acesta parcurge ruta contruită de la Node-ul țintă înapoi spre Node-ul de bază. La această etapă evenimentul este prelucrat de către manipulatoarele Node-urilor aflate pe această rută, dacă aceastea au fost aduagate.

Filtrele și manipulatoarele sunt o instanță a interfeței EventHandler<T extends Event>, ce conține metoda „*handle(T event)*”. În această metodă se descriu acțiunele care trebuiesc efectuate când evenimentul a ajuns la Node-ul pe care au fost setate filtrele, cu ajutorul metodei de bază „*addEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter*)”, sau manipulatoarele, cu ajutorul metodei de bază „*addEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)*” sau metodelor speciale.

Consumarea evenimentului se indică în medota „*handle(T event)*” folosind metoda generală din clasa javafx.event.Event „*consume()*”. Evenimentul fiind consumat nu mai pargurge ruta construită dintre Node-ul de bază și Node-ul țintă.

**II DESCRIEREA INSTRUMENTELOR SOFT UTILIZATE**

**2.1 IDE pentru scrierea codului**

IntelliJ IDEA este un IDE scris în Java folosit pentru a dezvolta software scrise în Java, Kotlin, Groovy și alte limbaje pe bazate pe JAR. Sunt două versiuni: Community Edition ce este gratis, dar limitat în limbajele și framework-urile folosite, și Ultimate ce este contra plată ce lipsit de limitările primei versiuni.

În acest instrumentar a fost scris toată partea logică a aplicației, adică codul în limbajul Java. La fel cu ajutorul acestuia a fost configurat automat Maven, necesar pentru stabilirea depedențelor în program.

Alternativ acestui IDE sunt Eclipse, NetBeans și Visual Studio Code cu extensiune pentru Java. Eclipse nu a fost în același mod de comod în utilizare. NetBeans consumă multă memorie în timpul folosirii sale. Visual Studio Code este un formator de text la care se poate de instalat extensiunea pentru Java, făcândul foarte limitat în funcționalitate.

S-a ales IntelliJ IDEA din motivul comodității și utilității sale. În acest IDE este încorporat completarea codului, integritatea cu GitHub, formatarea codului după placul utilizatorului și alte funcții care ușurează viața programatorului.

**2.2 Aplicația pentru aranjamentul părții grafice**

Scene Builder este un instrument în acces liber pentru toți folosit pentru a dezvolta interfața grafică pentru JavaFX folosind metoda trage și aruncare. Poate fi incorporat în IDE moderne, în cazul proiectului dat IntelliJ IDEA sau descărcat ca aplicație aparte.

Cu ajutorul acestui intrument a fost creată interfața grafică pentru aplicația, la care ulterior a fost conectată partea logică.

La momentul actual nu există concurent pentru acest instrument, de aceea alegerea a fost una ușoară.

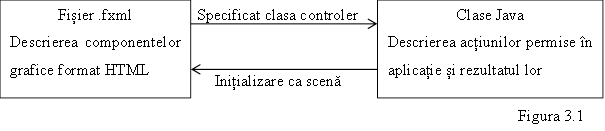
S-a ales Scene Builder pentru a nu crea interfața grafică prin cod, ci prin crearea, stilizarea și organizarea componentelor grafice folosind metoda tragere și aruncare. Alt motiv pentru alegerea acestuia a fost lipsa concurenților.

**III DESCRIEREA PĂRȚII PRACTICE**

Pentru a oferi o înțelegere mai bună a subiectului, ca temă pentru partea practică a fost ales o aplicație demonstrativă în care se analizează o parte din evenimente. Ca model au fost luate aplicațiile magazinelor virtuale. Produse concurente pot fi ignorate, din motivul scopului necesar de atins, adică instruire și înțelegere totală a tematicii.

Însăși aplicația poate fi descrisă în felul următor: posibilitatea de a alege un fișier executabil din memoria calculatorului și de al porni prin intermediul aplicației. Selectarea se face prin căutarea directă printre fișierele calculatorului sau prin tragere acestuia în zona specială din aplicație. Fișierele executabile se afișează într-un container cu iconița și butonul de pornire.

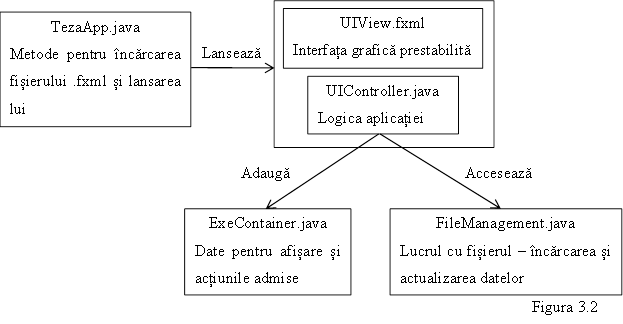
Aplicația este formată din 2 componente: partea grafică făcută cu ajutorul la JavaFX, păstrată într-un fișier cu extensia .fxml, și partea de cod scrisă în limbajul Java cu pachetele din JavaFX. Pentru efectuara conexiunii între componente este necesar în fișierul .fxml de specifact clasa Java răspunzătoare de comportamentul componentelor grafice, iar din partea de cod este necesar la lansarea aplicației de specificat să inițializeze fișierul .fxml și să-l seteze ca scenă.



În aplicație se prevede un singur utilizator – însăși persoana care folosește aplicația. Acțiunele acestuia afectează doar pe el însuți, calculatorul său și aplicația instalată pe el.

Partea de cod a fost împărțită în mai multe clase pentru a menține codul organizat și ușor de mențin. Au fost dezvoltate 4 clase: „TezaApp”, „FileManagement”, „ExeContainer”, „UIController” și o înregistrare: „ExeFile”. TezaApp răspunde de încărcarea fișierului .fxml, adică partea grafică prestabilită inițial și inițializarea aplicației. FileManagement răspunde de lucrul cu fișierul ce păstrează fișierele executabile adăugate în aplicație de către utilizator. ExeContainer este o componentă indepedentă cu afișarea sa și acțiunile permise asupra sa ce reprezintă fișiereul executabil în aplicație. UIController răspunde de partea logică a aplicației, adică citirea și răspunderea la acțiunile utilizatorului, folosirea de FileManagement și adăugarea în interfața grafică a ExeContainer. Iar ExeFile este o înregistrarea ce păstrează id-ul, numele și calea către fișierul executabil în memoria calculatorului folosită pentru transportul de date mai comod.

Partea grafică este reprezentată doar de un singur fișier „UIView.fxml”, ce conține datele despre componentele vizuale în aceeași formă de organizare ca limbajul de marcare HTML, adică folosirea de tag-uri în formă ierarhică. De asemenea în acest fișier se indică ce clasă va reprezenta clasa de control al componentelor grafice.



În așa mod aplicația dată corespunde unui tip de dezvoltare MVC, în care se prevede ca partea logică, vizuală și de control. Partea de control prea informația de la utilizator și le transformă în comenzi din partea logică. Partea logică se ocopă cu prelucrarea datelor și întoarcerea unui răspuns. Partea vizuală este o reprezentare grafică a răspunsului venit de la partea logică. Așa tip de model oferă o organizarea mai bună a codului, mai ales în aplicație de scară largă, și permite testarea mai eficientă a fiecărei părți. De asemenea este mai ușor de adăugat componente noi la cele existente fără a modifica esențial codul inițial.

**Concluzii**

**Bibliografie:**

1. <https://docs.oracle.com/javase/8/docs/api/java/util/EventObject.html>
2. <https://docs.oracle.com/javase/8/javafx/api/index.html>
3. <https://docs.oracle.com/javase/8/javafx/scene-graph-tutorial/scenegraph.htm>
4. <https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm>
5. <https://docs.oracle.com/javase/8/javafx/events-tutorial/events.htm>