**MINISTERUL EDUCAȚIEI și CERCETĂRII**

**UNIVERSITATEA TEHNICĂ a MOLDOVEI**

**FACULTATEA CALCULATOARE, INFORMATICĂ și MICROELECTRONICĂ**

**DEPARTAMENTUL INGINERIA SOFTWARE și AUTOMATICĂ**

**Proiect de an**

**Disciplina:**Tehnici și Mecanisme de Proiectare Software

**Tema:** Implementarea și proiectarea unui sistem pentru o aplicație de tip Mediaplayer

(Implementing and designing a system for a mediaplayer application)

|  |  |  |
| --- | --- | --- |
| **Student:** | **Bodrug Mihai** | **, TI-202** |
| **Coordonator:** | **Gaidau Mihai** | **, asis. univ.** |
|  |  |  |

**Chişinău, 2023**

**Cuprins**

[Introducere 2](#_Toc97828737)

[1 Analiza domeniului de studiu 3](#_Toc97828738)

[1.1 Scopul, obiectivele și cerințele sistemului 4](#_Toc97828739)

[1.2 Analiza sistemelor deja existente 5](#_Toc97828740)

[2. Realizarea sistemului 7](#_Toc97828741)

[2.1 Proiectarea aplicației 10](#_Toc97828742)

[2.2 Descrierea tehnologiilor pentru sistem 16](#_Toc97828743)

[2.3 Descrierea la nivel de cod pe module 17](#_Toc97828744)

[3. Documentarea produsul realizat 22](#_Toc97828745)

[Concluzii 24](#_Toc97828746)

[Bibliografie 25](#_Toc97828747)

[Anexa A 26](#_Toc97828748)

# Introducere

Un media player (în traducere: aparat care redă medii) este un termen care semnifica un program de calculator / software care creează și redă înregistrări (fișiere) audio-vizuale. Este vorba de un aparat virtual, care poate prezenta pe monitorul calculatorului, printre altele, simbolurile uzuale pentru butoanele de start ►, stop ■, pauză ║, derulare rapidă înapoi ◄◄ ș.a.m.d., care pot fi activate („apăsate”) cu ajutorul mouse-ului.

Lucrarea dată urmărește scopul de a elabora crearea unei aplicații simple de tip mediaplayer cu un funcțional de baza ce ar permite redarea unui fișier video.

Lucrarea este structurată în trei capitole, unde se va analiza concret modul de proiectare și de realizare a proiectului dat, tehnologiile utilizate dar si modul de realizare a unei astfel de aplicații.

Primul dintre aceste capitole vine să facă o introduce generala asupra proiectului, el cuprinde scopul, obiectivele și cerințele sistemului dat. Totodată in acest capitol se face analiza domeniului de studiu si a sistemelor deja existe de acest tip.

Al doilea capitol cuprinde partea de realizare a acestui sistem, in acest capitol are loc realizarea la nivel de cod a aplicației si implementarea șabloanelor de proiectare in acest proces. Aplicația data are sa fie proiectata iar tehnologiile utilizate sa fie descrise. Descrierea codului se va face pe module, deosebita atenție fiind acordata șabloanelor de proiectare utilizate.

Acest capitol este unul ce tine mai mult de lucrul practic însă acest lucru are sa conțină o documentație relevanta.

În ultimul capitol se vorbește despre aplicație la general si se face o documentație a produsului realizat. Acest capitol fiind unul de informare despre modul de funcționare a produsului. Elementele de documentație a aplicației sunt: denumirea, caracteristicile, imagini sau descrierea. La sfârșit se va efectua concluzia acestui proiect unde se va explica treptat realizarea acestei aplicații, demonstrarea importantei unui astfel de aplicații in domeniul ales, explicarea impactului unei astfel de aplicații.

# Analiza domeniului de studiu

Aplicația face parte din domeniul tehnologiilor informaționale. Aplicația este de tip desktop având un GUI ce permite activitățile de baza. Astfel de aplicații sunt utilizate in viața cotidiana pentru activități simple(vizionarea unui film, ascultarea unui cântec) însă fără de aceste aplicații ar apărea unele dificultăți.

Șabloanele de proiectare sunt un set de instrumente pentru soluții încercate la probleme comune în proiectarea software. Chiar dacă nu întâlniți niciodată aceste probleme, cunoașterea șabloanelor este încă utilă, deoarece vă învață cum să rezolvați tot felul de probleme folosind principii de design orientat pe obiecte.[[1]](https://www.studocu.com/ro/document/universitatea-de-vest-din-timisoara/design-patterns/sabloane-de-proiectare/8021574)

Șabloanele de proiectare diferă prin complexitatea lor, nivelul de detaliu și scala de aplicabilitate la întregul sistem proiectat. Îmi place analogia cu construcția drumurilor: puteți face o intersecție mai sigură fie instalând niște semafoare sau construind o întreagă intersecție amenajată pe mai multe niveluri cu pasaje subterane pentru pietoni. Cele mai de bază și de niveluri scăzute șabloane sunt adesea numite idiomuri. De obicei, se aplică numai unui singur limbaj de programare. Cele mai universale și de niveluri înalte sunt modele arhitecturale. Dezvoltatorii pot implementa aceste tipare în aproape orice limbă. Spre deosebire de alte șabloane ,ele pot fi utilizate pentru a proiectate arhitectura unei aplicații întregi. În plus, toate tiparele pot fi clasificate în funcție de intenția sau scopul lor. Această carte acoperă trei grupuri principale de modele:

* Șabloanele de creație oferă mecanisme de creare a obiectelor care cresc flexibilitatea și reutilizarea codului existent.
* Șabloanele structurale explică modul de asamblare a obiectelor și claselor în structuri mai mari, păstrând în același timp structurile flexibile și eficiente.
* Șabloanele de comportament au grijă de o comunicare eficientă și de atribuirea responsabilităților între obiecte. [[1]](https://www.studocu.com/ro/document/universitatea-de-vest-din-timisoara/design-patterns/sabloane-de-proiectare/8021574)

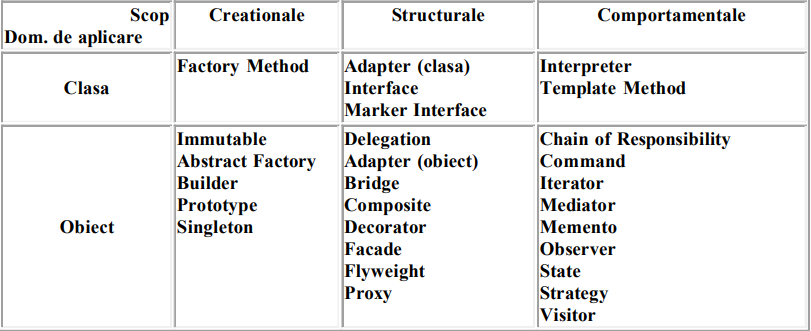


Fig.1 Clasificarea șabloanelor[[2]](http://inf.ucv.ro/~mihaiug/courses/poo/slides/Curs%2012%20-%20Sabloane%20de%20Proiectare.pdf)

In Fig.1 Clasificarea șabloanelorsunt reprezentate cele mai importante șabloane de proiectare la momentul actual.

# 1.1 Scopul, obiectivele și cerințele sistemului

Principalul scop al acestui proiect este crearea unei aplicații de tip Mediaplayer cu un funcțional de baza minim. Design-ul va fi minimalist pentru a nu încarcă sistema. Aplicația v-a putea funcționa cu extensii a file-urilor de tip .mp4. Player-ul va deține butoane de tip Play, Stop, Pause, Sound slide. Culorile vor fi de baza, nu va fi posibil selectarea unei teme pentru aplicație (Dark, Light). Aplicația nu va deține o baza de date, selectarea file-urilor pentru derulare fiind directa.

Obiectivele de baza sunt:

* Realizarea unei aplicații de tip mediaplayer;
* Utilizarea șabloanelor de proiectare;
* Utilizarea minim a 2 șabloane de proiectare;
* Respectarea principiilor SOLID.

Cerințele sistemului sunt:

* Aplicație simpla;
* Consum de memorie minim;
* Posibilitatea redării fișierelor media;
* Posibilitatea schimbării volumului;
* Prezenta butoanelor de Pause si Play.



Fig.2 Interfața GUI

In Fig.2 Interfața GUIeste reprezentata GUI unei aplicații simple de tip mediaplayer care este ușor de implementat.

# 1.2 Analiza sistemelor deja existente

In urma analizei sistemelor existente la momentul actual am gasit niste sisteme asemănătoare cu sistemul dat:

* Windows Media Player;
* Media Player Classic;
* VLC.

Windows Media Player este aplicația implicita din Windows care permite redarea unui video. Aplicația are un GUI simplu dar cu un funcțional bogat. Cântecele pot fi sortate după criterii, pot fi create playlist-uri după dorința si nu in ultimul rând pot fi afișate poze in loc de file-uri video. In Fig.3 Windows Media Player Classic am reprezentat Windows Media Player.

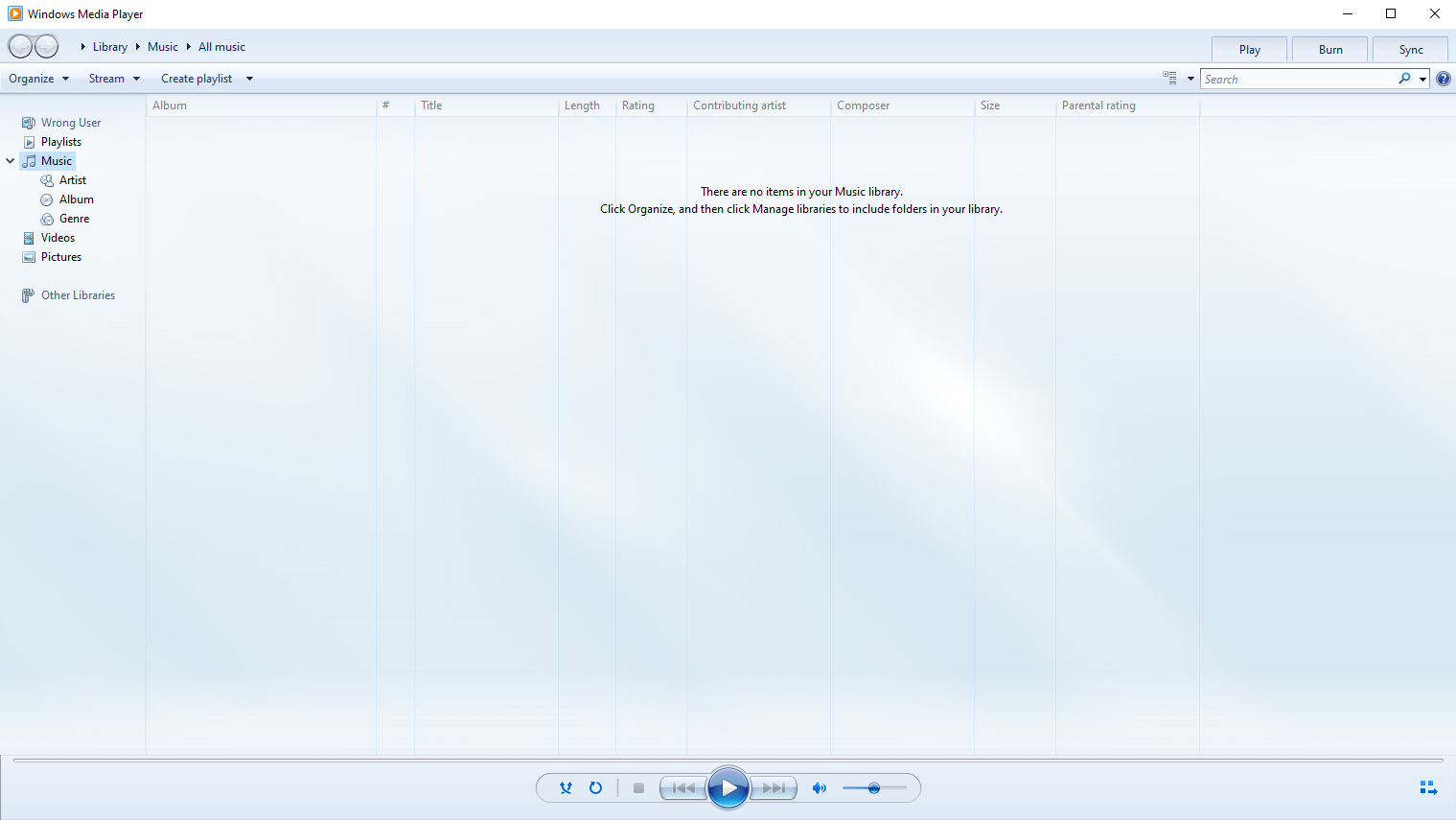


Fig.3 Windows Media Player Classic

Media Player Classic este un player media gratuit si open-source, ușor personalizabile pentru Windows pe 32 si 64 biti. Lansare finala a avut loc in 2015. La moment acest proiect este întreținut in principal de comunitatea de pe forum-uri. Cum arata fereastra aplicației se poate de observat in Fig.4 Media Player Classic.



Fig.4 Media Player Classic

VLC Media Player este un mediaplayer liber si open source. Aplicația este portabila si cross-platform. Deține un funcțional destul de bogat însă interfața GUI nu a suportat schimbări drastice. Este posibila instalarea unor extensii adiționale in cazul in care este nevoie. Instanța aplicației are o interfața ușor de înțeles si intuitiva, aceasta se poate observa in Fig.5 VLC Media Player.

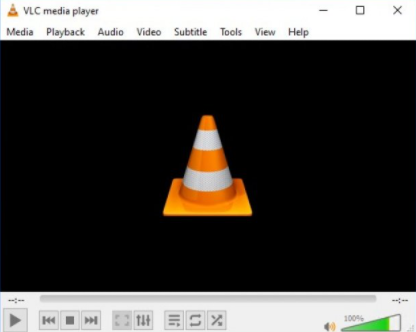


Fig.5 VLC Media Player

Toate aceste sisteme au in comun funcționalul simplu dar comod si ușor de înțeles. Ele dețin un GUI modern, funcțional extins pentru diferite necesitați, posibilitatea lucrului cu file-uli cu diferite extensii.

# 2. Realizarea sistemului

Realizarea sistemului a început cu structurarea aplicației si implementarea șabloanelor de proiectarea la nivel de cod. In total au fost implementate 6 șabloane, acestea fiind:

* Singleton;
* Builder;
* Decorator;
* Adapter;
* Observer;
* Memento.

Fiecare din aceste șabloane are ca funcție rezolvarea unei probleme sau îmbunătățirea funcționalului aplicației prin utilizarea unor metode existente ce ar preveni unele probleme.

Șablonul Singleton este un șablon de proiectare software care restricționează instanțierea unei clase la o singură instanță. Acest lucru este util atunci când este necesar un singur obiect pentru a coordona acțiunile în sistem. Adesea acest șablon se utilizează in următoarele cazuri:

* Abstract Factory, Factory method, Builder și Prototype pot folosi singleton-uri;
* Obiectele de Facade sunt adesea unice, deoarece este necesar un singur obiect de facade;
* Obiectele statice sunt adesea unice;

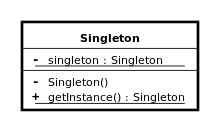


Fig.6 Diagrama de clasa a șablonului Singleton

Diagrama de clasa a șablonului Singleton este reprezentata in Fig.6 Diagrama de clasa a șablonului Singleton

Șablonul Builder este un șablon de proiectare conceput pentru a oferi o soluție flexibilă la diferite probleme de creare a obiectelor în programarea orientată pe obiecte. Intenția modelului de design Builder este de a separa construcția unui obiect complex de reprezentarea acestuia. Este unul dintre modelele de design Gang of Four.

Diagrama de clasa a acestui șablon este reprezentata in Fig.7 Diagrama de clasa a șablonului Builder.

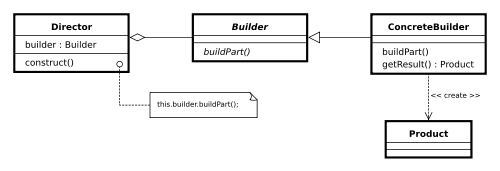


Fig.7 Diagrama de clasa a șablonului Builder.

Șablonul decorator este un șablon de design care permite adăugarea comportamentului unui obiect individual, în mod dinamic, fără a afecta comportamentul altor obiecte din aceeași clasă. Șablonul Decorator este adesea util pentru aderarea la Principiul responsabilității unice, deoarece permite ca funcționalitatea să fie împărțită între clase cu domenii unice de interes. Utilizarea decoratorului poate fi mai eficientă decât subclasarea, deoarece comportamentul unui obiect poate fi mărit fără a defini un obiect complet nou. Diagrama de clasa a șablonului dat este indicat in Fig.8 Diagrama de clasa a șablonului Decorator.



Fig.8 Diagrama de clasa a șablonului Decorator

Șablonul Adapter este un șablon de proiectare software (cunoscut și sub numele de wrapper, o denumire alternativă partajată cu modelul decorator) care permite ca interfața unei clase existente să fie utilizată ca o altă interfață. Este adesea folosit pentru a face ca clasele existente să funcționeze cu alții fără a modifica codul sursă. Diagrama de clasa a șablonului dat este indicat in Fig.9 Diagrama de clasa a șablonului Adapter.

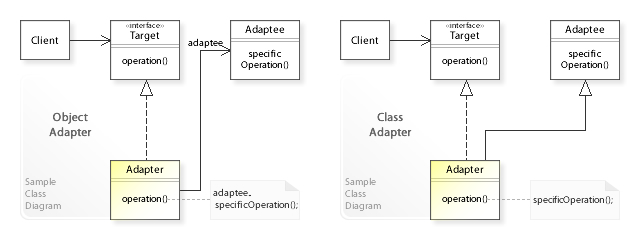


Fig.9 Diagrama de clasa a șablonului Adapter.

Șablonul Observer este un șablon de proiectare software în care un obiect, numit subiect, menține o listă a dependenților săi, numiți observatori, și îi notifică automat despre orice schimbare de stare, de obicei apelând una dintre metodele lor.

Este folosit în principal pentru implementarea sistemelor distribuite de gestionare a evenimentelor, în software-ul „event driven”. În aceste sisteme, subiectul este de obicei denumit „flux de evenimente” sau „sursă de flux de evenimente”, în timp ce observatorii sunt numiți „stream de evenimente”. Nomenclatura fluxului face aluzie la o configurație fizică în care observatorii sunt separați fizic și nu au control asupra evenimentelor emise de subiectul/sursa fluxului. Acest model se potrivește perfect oricărui proces în care datele sosesc de la o intrare care nu este disponibilă pentru procesor la pornire, ci în schimb sosesc „la întâmplare” (cereri HTTP, date GPIO, intrare de utilizator de la tastatură/mouse/..., baze de date distribuite). și blockchains, ...). Majoritatea limbajelor de programare moderne includ constructe „eveniment” încorporate care implementează componentele modelului de observator. Diagrama de clasa a acestui șablon este reprezentata in Fig.10 Diagrama de clasa a șablonului Observer.

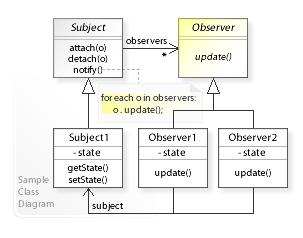


Fig.10 Diagrama de clasa a șablonului Observer.

Ultimul șablon implementat a fost Șablonul Memento. Șablonul memento este un șablon de proiectare software care oferă capacitatea de a restabili un obiect la starea anterioară (anulare prin rollback).

Șablonul memento este implementat cu trei obiecte: inițiatorul, un îngrijitor și un memento. Inițiatorul este un obiect care are o stare internă. Îngrijitorul îi va face ceva inițiatorului, dar vrea să poată anula schimbarea. Îngrijitorul cere mai întâi inițiatorului un obiect memento. Apoi face orice operație (sau secvență de operații) pe care urma să o facă. Pentru a reveni la starea dinaintea operațiunilor, returnează obiectul memento inițiatorului. Obiectul memento în sine este un obiect opac (unul pe care îngrijitorul nu poate sau nu ar trebui să îl schimbe). Când utilizați acest model, trebuie să aveți grijă dacă inițiatorul poate schimba alte obiecte sau resurse - modelul memento operează pe un singur obiect. Diagrama de clasa este reprezentată in Fig.11 Diagrama de clasa a șablonului Memento.

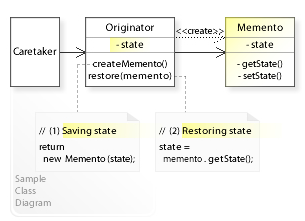


Fig.11 Diagrama de clasa a șablonului Memento.

# 2.1 Proiectarea aplicației

Prin modelarea conceptuală a datelor (analiza si definirea cerințelor) se urmărește construirea unui model al datelor care să asigure transpunerea exactă a realității din domeniul analizat, fără a lua în considerare cerințele specifice unui model de organizare a datelor (cum este modelul relațional), criteriile de calitate privind organizarea datelor, cerințelor nefuncționale ale sistemului şi criteriile de performanta privind stocarea și accesarea datelor.

In figura 12 este reprezentata diagrama interacțiunilor utilizatorului cu sistemul dat. Utilizatorul poate interacționa cu sistemul dat prin mai multe metode.

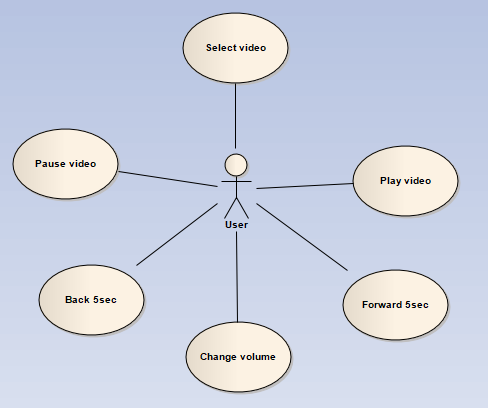


Figura 12. Interacțiunea user-ului cu sistemul

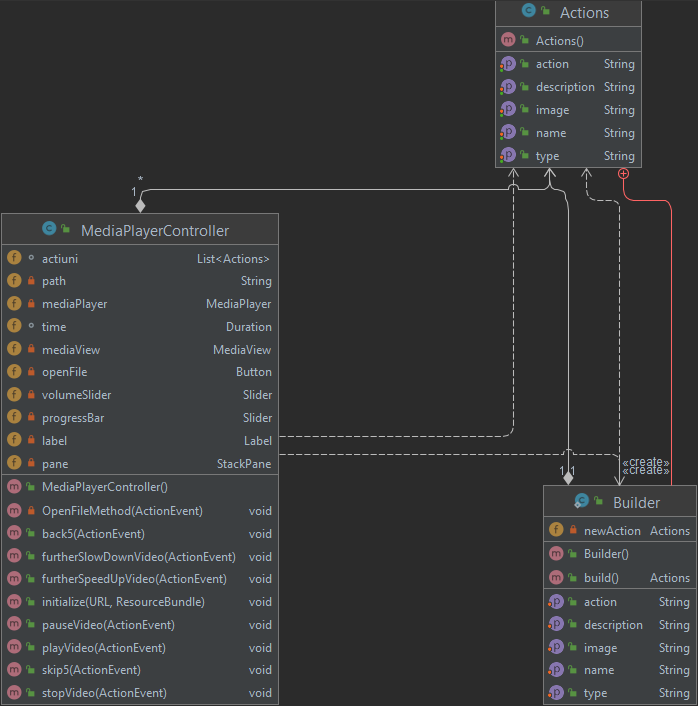


Figura 12.1 Diagrama de clasă a șablonului Builder

In Figura 12.1 Diagrama de clasă a șablonului Builder este reprezentat modul de funcționare a șablonului Builder in aceasta aplicație. El permite crearea unei acțiuni care consta din mai multe elemente (name, type, description etc). Cu ajutorul acestor acțiuni poate fi creata o întreaga istorie a file-urilor media adăugate.

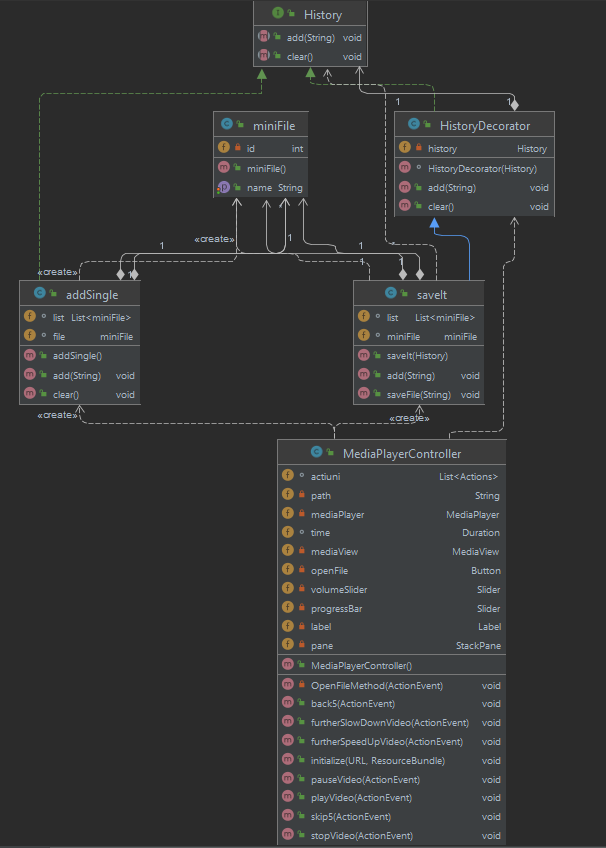


Figura 12.2 Diagrama de clasa a sablonului Decorator

Șablonul Decorator in aceasta aplicație ne permite adăugarea unui funcțional adițional unei funcții deja existente. Cu ajutorul acestui șablon funcției de notificare a faptului ca un fișier a fost adăugat i s-a adăugat funcționalul de a salva acest file cu numele său respectiv. Diagrama de clasa a acestui șablon este reprezentata in Figura 12.2 Diagrama de clasa a șablonului Decorator.

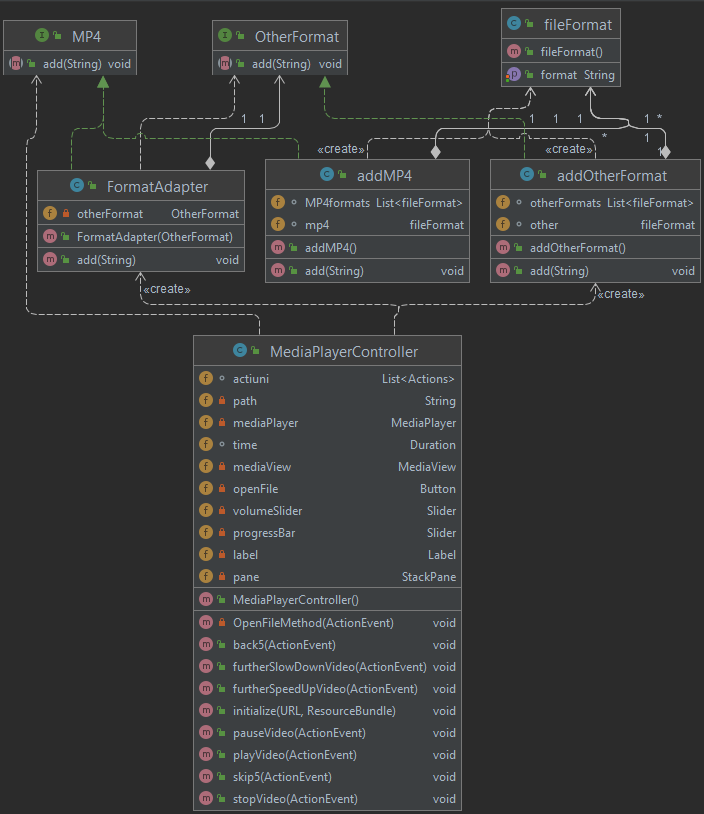


Figura 12.3 Diagrama de clasă a șablonului Adapter

Șablonul Adapter permite ca interfața unei clase existente să fie utilizată ca o altă interfață. Astfel interfața MP4 este utilizata de către clasa addOtherFormat, fiecare file nou adăugat fiind interpretat ca un file .mp4 si tratat respectiv. Modul de funcționare a acestui șablon in aplicație este reprezentat in Figura 12.3 Diagrama de clasa a șablonului Adapter.

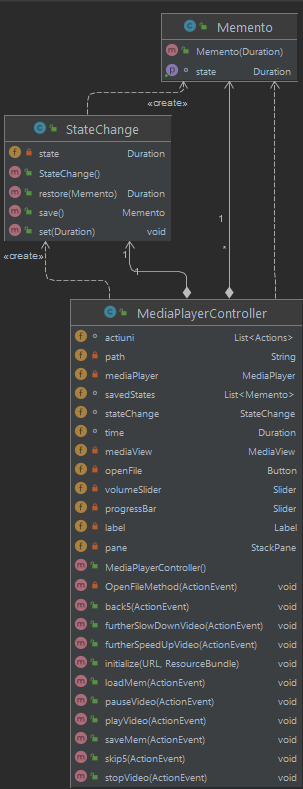


Figura 12.4 Diagrama de clasa a sablonului Memento

Șablonul Memento in aceasta aplicație ne permite implementarea unui sistem de marcaje de timp (timestamp). Cu ajutorul acestei funcționalități orice moment in timpul redării unui fișier poate fi salvat si se poate întoarce la el. Diagrama acestui șablon este arătata in Figura 12.4 Diagrama de clasa a șablonului Memento

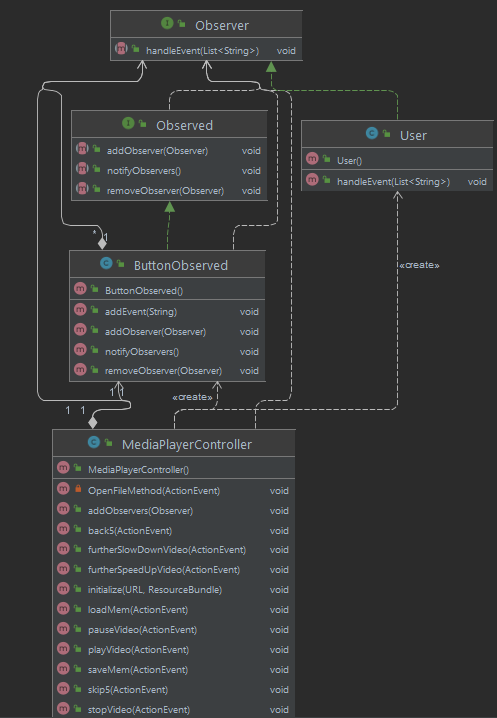


Figura 12.5 Diagrama de clasa a sablonului Observer

Ultimul șablon implementat este Observer. Observer menține o listă a dependenților săi, numiți observatori, și îi notifică automat despre orice schimbare de stare, de obicei apelând una dintre metodele lor. In cazul dat când este apăsat un buton ce schimbă starea sistemului( Play, pause, stop) utilizatorul este notificat. Diagrama de clasa a acestui șablon este reprezentata in Figura 12.5 Diagrama de clasa a șablonului Observer.

# 2.2 Descrierea tehnologiilor pentru sistem

Pentru realizarea acestei aplicații a fost ales limbajul Java. Toata aplicatia fiind scrisa in Java.

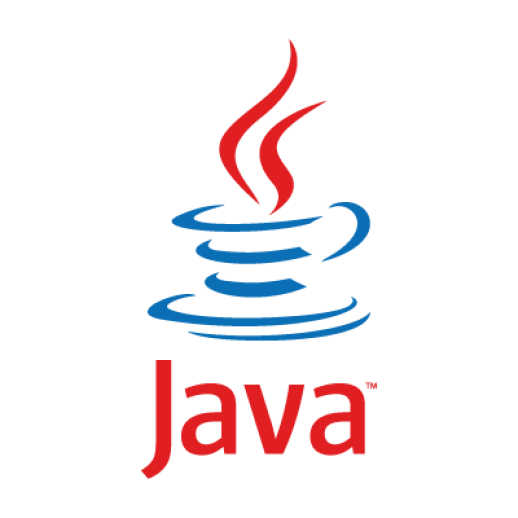


Fig.13 Logotipul java

Principala librărie utilizata in elaborarea acestei aplicatii a fost JavaFX. JavaFX este o platformă bazată pe Java pentru construirea de aplicații GUI. Poate fi folosit atât pentru a crea aplicații desktop care rulează direct din sistemele de operare, cât și pentru aplicații de Internet ( RIA ) care rulează în browsere , cât și pentru aplicații pe dispozitive mobile



Fig.14 Logotipul JavaFX

Pentru crearea file-ului de tip .fxml(partea vizuala) a fost utilizat Scene Builder

****

Fig.14 Logotipul SceneBuilder

# 2.3 Descrierea la nivel de cod pe module

Sablonul Singleton:

Se verifica daca este doar o singura instanta a aplicatiei.

private static MediaPlayer instance;  
  
public static synchronized MediaPlayer getInstance() {  
 if (instance == null) {  
 instance = new MediaPlayer();  
 }  
 return instance;  
}

Sablonul Builder:

Cu ajutorul sablonului Builder este ,,construita” o actiune ce contine nume, tip, descriere.

public static class Builder{  
 private final Actions newAction;  
  
 public Builder(){  
  
 newAction = new Actions();  
 }  
  
 public Builder setName(String name){  
 newAction.setName(name);  
 return this;  
 }  
  
 public Builder setType(String type){  
 newAction.setType(type);  
 return this;  
 }  
  
 public Builder setDescription(String description){  
 newAction.setDescription(description);  
 return this;  
 }  
  
 public Builder setImage(String image){  
 newAction.setImage(image);  
 return this;  
 }  
  
 public Builder setAction(String action){  
 newAction.setAction(action);  
 return this;  
 }  
  
 public Actions build(){  
 return newAction;  
 }  
}

Sablonul Adapter:

Adapter permite ca interfața unei clase existente să fie utilizată ca o altă interfață. Astfel interfața MP4 este utilizata de către clasa addOtherFormat, fiecare file nou adăugat fiind interpretat ca un file .mp4 si tratat respectiv.

Avem 2 interfete principale MP4 si OtherFormat:

public interface MP4 {  
 void add(String filename);  
}

public interface OtherFormat {  
 void add(String filename);  
}

Clasele ce implementeaza aceste interfete:

public class addMP4 implements MP4{  
 List<fileFormat> MP4formats = new ArrayList<>();  
 fileFormat mp4 = new fileFormat();  
 @Override  
 public void add(String filename) {  
 mp4.setFormat(filename);  
 MP4formats.add(mp4);  
 for (fileFormat f : MP4formats) {  
 System.out.println(f.getFormat());  
 }  
  
 }  
}

public class addOtherFormat implements OtherFormat{  
 List<fileFormat> otherFormats = new ArrayList<>();  
 fileFormat other = new fileFormat();  
 @Override  
 public void add(String filename) {  
 other.setFormat(filename);  
 otherFormats.add(other);  
 System.out.println("Opened a "+filename+" file");  
 }  
}

Clasa adapter:

public class FormatAdapter implements MP4{  
 private final OtherFormat otherFormat;  
 public FormatAdapter(OtherFormat otherFormat){  
 this.otherFormat = otherFormat;  
 }  
 @Override  
 public void add(String filename) {  
 otherFormat.add(filename);  
 }  
}

Sablonul Decorator:

Șablonul Decorator permite adăugarea unui funcțional adițional unei funcții deja existente. Cu ajutorul acestui șablon funcției de notificare a faptului ca un fișier a fost adăugat i s-a adăugat funcționalul de a salva acest file cu numele său respectiv.

public interface History {  
 void add(String s);  
 void clear();  
}

public class addSingle implements History {  
 List<miniFile> list = new ArrayList<>();  
 miniFile file = new miniFile();  
 @Override  
 public void add(String s) {  
 file.setName(s);  
 list.add(file);  
 System.out.println("Opened a file");  
  
 }  
  
 @Override  
 public void clear() {  
 list.clear();  
 System.out.println("Cleared");  
 }  
}

public class HistoryDecorator implements History{  
 private final History history;  
 HistoryDecorator(History history){  
 this.history = history;  
 }  
 @Override  
 public void add(String s) {  
 history.add(s);  
 }  
 @Override  
 public void clear() {  
 history.clear();  
 }  
}

public class saveIt extends HistoryDecorator{  
 List<miniFile> list = new ArrayList<>();  
 miniFile miniFile = new miniFile();  
  
 public saveIt(History h){  
 super(h);  
  
 }  
  
 public void saveFile(String name){  
 miniFile.setName(name);  
 list.add(miniFile);  
 for(miniFile m:list){  
 System.out.println(m.getName());  
 }  
 }  
  
 @Override  
 public void add(String s) {  
 super.add(s);  
 saveFile(s);  
 System.out.println("saved");  
  
 }  
}

Sablonul Observer:

Observer permite instiintarea tuturor observatorilor despre schimbarea care a avut loc cu obiectul observat, aceasta instiintare are loc prin chemarea unei functii.

public interface Observer {  
 void handleEvent(List<String> event);  
}

public interface Observed {  
 public void addObserver(Observer o);  
 public void notifyObservers();  
}

public class User implements Observer{  
 @FXML  
 private Slider volumeSlider;  
 @Override  
 public void handleEvent(List<String> event) {  
 switch(event.get(event.size() - 1)){  
 case "play":  
 System.out.println("User: click play");  
 break;  
 case "pause":  
 System.out.println("User: click pause");  
 break;  
 case "stop":  
 System.out.println("User: click stop");  
 break;  
 }  
 //System.out.println("Video has changed state: "+event+"\n");  
 }  
}

public class ButtonObserved implements Observed{  
  
 List<String> event = new ArrayList<>();  
 List<Observer> observers = new ArrayList<>();  
  
 public void addEvent(String event){  
 this.event.add(event);  
 notifyObservers();  
 }  
  
  
 @Override  
 public void addObserver(Observer o) {  
 this.observers.add(o);  
 }  
  
 @Override  
 public void removeObserver(Observer o) {  
 this.observers.remove(o);  
 }  
  
 @Override  
 public void notifyObservers() {  
 for (Observer observer: observers){  
 observer.handleEvent(this.event);  
 }  
 }  
}

Sablonul Memento:

Sablonul Memento permite implementarea unui sistem de marcaje de timp (timestamp). Cu ajutorul acestei funcționalități orice moment in timpul redării unui fișier poate fi salvat si se poate întoarce la el. El consta din 2 clase(Memento si StateChange).

public class Memento {  
 private final Duration state;  
 public Memento(Duration state){  
 this.state = state;  
 }  
 Duration getState(){  
 return state;  
 }  
}

public class StateChange {  
 private Duration state;  
 public void set(Duration state) {  
 this.state = state;  
 }  
 public Memento save() {  
 return new Memento(this.state);  
 }  
 public Duration restore(Memento memento) {  
 this.state = memento.getState();  
 return state;  
 }  
}

# 3. Documentarea produsul realizat

După deschiderea aplicației prin efectuarea unui dublu click se deschide interfața principala care arata in modul reprezentat în Fig.15 Interfața Principala.

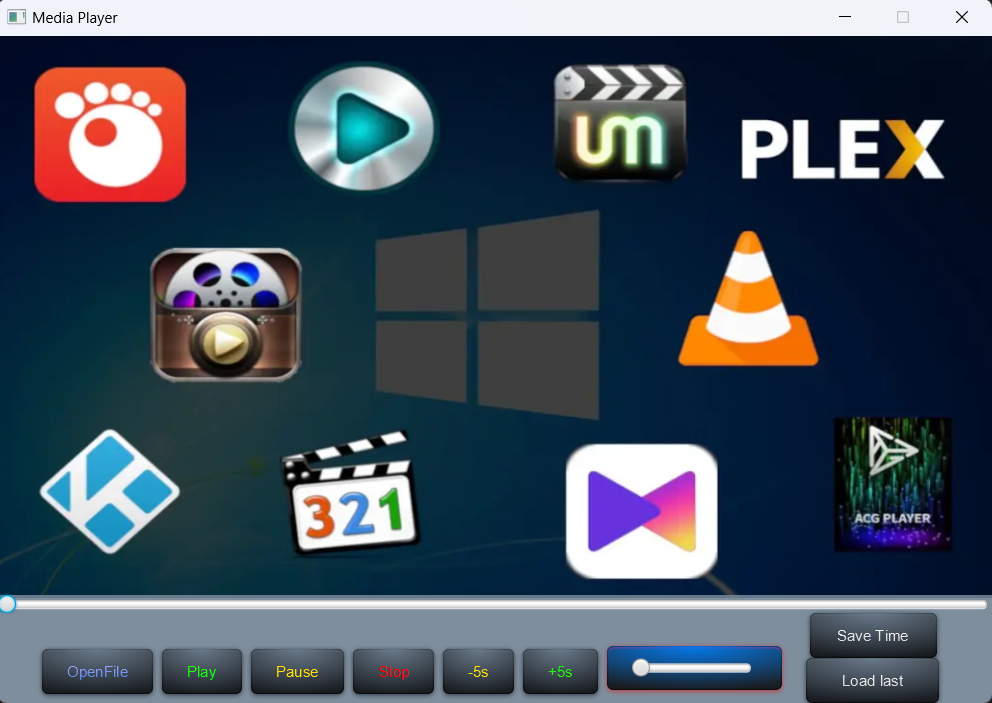


Fig.15 Interfața Principala

După accesarea butonului indicat în Fig.15.1 Butonul OpenFile se selectează un file de pe PC cu ajutorul butonului OpenFile. Fișierul urmează sa fie de tip mp3, mp4, wav.



Fig.15.1 Butonul OpenFile

După selectarea file-ului se apasă butonul Play pentru rularea fișierului. Modul în care arata butonul este reprezentat în Fig.15.2 Butonul Play.



Fig.15.2 Butonul Play

In timpul rulării poate fi modificat volumul cu ajutorul slide-urului. Slider-ul se afla in partea dreapta a aplicației. Slider-ul dat este arătat în Fig.15.3 Slider-ul volumului



Fig. 15.3 Slider-ul volumului

Butonul Pause setează pauza iar butonul Stop oprește definitiv rularea file-ului. Ambele butoane sunt reprezentate respectiv în Fig.15.4 Butonul Pause si Fig.15.5 Butonul Stop.



Fig.15.4 Butonul Pause



Fig.15.5 Butonul Stop

La efectuarea unui click pe butonul Save indicat în Fig.15.6 Butonul Save Time se creează un timestamp la care se poate reveni ulterior în caz de nevoie.



Fig.15.6 Butonul Save Time

Pentru revenirea la un timestamp care a fost salvat mai înainte poate fi accesat butonul Load Time indicat în face load la ultimul timestamp salvat



Fig.15.7 Butonul Load Time

# Concluzii

Șabloanele de proiectare sunt niște soluții generale care pot rezolva unele probleme ce apar la realizarea unei aplicații complexe. In acest proiect de an am realizat o aplicație ce utilizează 6 șabloane de proiectare diferite. Șabloanele implementate au fost de 3 tipuri: structurale, comportamentale si creațiunile.

Șabloanele creațiunile utilizate au fost Builder si Singleton. Șabloanele structurale implementate sunt Decorator si Adapter. Ultimele 2 șabloane sunt de tipul șabloanelor comportamentale si anume Memento si Observer.

Utilizarea șabloanelor permite optimizarea codului si permite ca codul sa fie ușor de modificat in viitor in cazul adăugării unui funcțional nou din cauza ca șabloanele utilizează principiile SOLID, datorita acestui fact adăugarea unor funcții noi nu ar trebui sa creeze dificultăți sau probleme majore la refactor codului.

O astfel de aplicație poate rula perfect cu aceste 6 șabloane, fiecare șablon care a fost implementat are o funcție bine definita si un rol important. Deși rezultatul lucrului unui șablon nu este mereu vizual vizibil este poate fi urmat in timpul funcționarii aplicației prin utilizarea unor acțiuni adăugătoare.

Schemele UML s-au primit complexe însă cu ajutorul lor se poate înțelege ce are loc in aplicație si ce interacțiune se petrec intre componentele acestei aplicații. Fiecare schema arata interacțiunile specifice unui șablon si cum acesta interacționează cu restul aplicației.

Scopul propus de aceasta lucrare a fost de a aduce la cunostinta principiile de functionare a sablonelor de proiectare si modul lor de implimentare în scopuri proprii. De asemeni sper ca prin lucrarea data o persoana va putea sa obtina o idee si informatie mai larga privind sabloanele de proiectare

# Bibliografie

1. Sabloane de proiectare; [Resursă electornică.] – Regim de acces:  
   https://www.studocu.com/ro/document/universitatea-de-vest-din-timisoara/design-patterns/sabloane-de-proiectare/8021574
2. Sabloane de proiectare. Design Patterns; [Resursă electornică.] – Regim de acces:  
   http://inf.ucv.ro/~mihaiug/courses/poo/slides/Curs%2012%20%20Sabloane%20de%20Proiectare.pdf
3. Java [Resursa electronică.] – Regim de acces: <https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)>
4. JavaFX [Resursa electronică.] – Regim de acces:

<https://ro.frwiki.wiki/wiki/JavaFXH>

1. SceneBuilder [Resursa electronică.] – Regim de acces:

<https://fr.wikipedia.org/wiki/Scene_Builder>

1. Learning SOLID Principles [Resursa electronică.] – Regim de acces:

<https://riptutorial.com/Download/solid-principles.pdf>

1. Memento in Java [Resursa electronică.] – Regim de acces:

<https://refactoring.guru/design-patterns/memento/java/example>

**Codul sursa al fisierului MediaPlayer**

package mediaplayer;  
  
import javafx.application.Application;  
import javafx.event.EventHandler;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.input.MouseEvent;  
import javafx.stage.Stage;  
public class MediaPlayer extends Application {  
  
 public enum EnumSingleton {  
  
 INSTANCE("Initial class info");  
  
 private String info;  
  
 EnumSingleton(String info) {  
 this.info = info;  
 }  
  
 public EnumSingleton getInstance() {  
 return INSTANCE;  
 }  
  
 // getters and setters  
 public String getInfo() {  
 return info;  
 }  
  
 public void setInfo(String info) {  
 this.info = info;  
 }  
  
 }  
  
 @Override  
 public void start(Stage stage) throws Exception {  
 Parent root = FXMLLoader.load(getClass().getResource("MediaPlayer.fxml"));  
  
 Scene scene = new Scene(root);  
 stage.setTitle("Media Player");  
  
 scene.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent event) {  
 if(event.getClickCount() ==2){  
 stage.setFullScreen(true);  
 }  
 }  
 });  
  
  
  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 EnumSingleton singleton = EnumSingleton.INSTANCE.getInstance();  
 singleton.setInfo("Launched");  
 System.out.println(singleton.getInfo());  
  
 launch(args);  
  
 }  
}

**Codul sursa al fisierului MediaPlayerController**

package mediaplayer;  
  
import java.io.File;  
import java.net.URL;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.ResourceBundle;  
import javafx.beans.InvalidationListener;  
import javafx.beans.Observable;  
import javafx.beans.binding.Bindings;  
import javafx.beans.property.DoubleProperty;  
import javafx.beans.value.ChangeListener;  
import javafx.beans.value.ObservableValue;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.fxml.FXML;  
import javafx.fxml.Initializable;  
import javafx.scene.control.Label;  
import javafx.scene.media.Media;  
import javafx.scene.media.MediaPlayer;  
import javafx.scene.media.MediaView;  
import javafx.stage.FileChooser;  
import javafx.scene.control.Button;  
import javafx.scene.control.Slider;  
import javafx.scene.input.MouseEvent;  
import javafx.scene.layout.StackPane;  
import mediaplayer.adapter.FormatAdapter;  
import mediaplayer.adapter.MP4;  
import mediaplayer.adapter.addMP4;  
import mediaplayer.adapter.addOtherFormat;  
import mediaplayer.builder.Actions;  
import mediaplayer.decorator.HistoryDecorator;  
import mediaplayer.decorator.addSingle;  
import mediaplayer.decorator.saveIt;  
  
  
public class MediaPlayerController implements Initializable {  
  
List<Actions> actiuni = new ArrayList<>();  
 private String path;  
 private MediaPlayer mediaPlayer;  
  
 @FXML  
 private MediaView mediaView;  
  
 @FXML  
 private Button openFile;  
  
 @FXML  
 private Slider volumeSlider;  
  
 @FXML  
 private Slider progressBar;  
  
 @FXML  
 private Label label;  
  
 @FXML  
 private StackPane pane;  
  
 @FXML  
 private void OpenFileMethod(ActionEvent event) {  
 FileChooser fileChooser = new FileChooser();  
 File file = fileChooser.showOpenDialog(null);  
 path = file.toURI().toString();  
  
  
 if(path != null){  
 Media media = new Media(path);  
 mediaPlayer = new MediaPlayer(media);  
 mediaView.setMediaPlayer(mediaPlayer);  
  
 // System.out.println(file.getName());  
 String extens = getExtension.getFileExtension(file.getName());  
 MP4 mp4 = new FormatAdapter(new addOtherFormat());  
 mp4.add(extens);  
  
 DoubleProperty widthProp = mediaView.fitWidthProperty();  
 DoubleProperty heightProp = mediaView.fitHeightProperty();  
  
 widthProp.bind(Bindings.selectDouble(mediaView.sceneProperty(), "width"));  
 heightProp.bind(Bindings.selectDouble(mediaView.sceneProperty(), "height"));  
  
 volumeSlider.setValue(mediaPlayer.getVolume()\*100);  
 volumeSlider.valueProperty().addListener(new InvalidationListener() {  
 @Override  
 public void invalidated(Observable observable) {  
 mediaPlayer.setVolume(volumeSlider.getValue()/100);  
 }  
 });  
  
 mediaPlayer.currentTimeProperty().addListener(new ChangeListener<javafx.util.Duration>() {  
 @Override  
 public void changed(ObservableValue<? extends javafx.util.Duration> observable, javafx.util.Duration oldValue, javafx.util.Duration newValue) {  
 progressBar.setValue(newValue.toSeconds());  
 }  
 }  
 );  
  
 progressBar.setOnMousePressed(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent event) {  
 mediaPlayer.seek(javafx.util.Duration.seconds(progressBar.getValue()));  
 }  
 });  
  
 progressBar.setOnMouseDragged(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent event) {  
 mediaPlayer.seek(javafx.util.Duration.seconds(progressBar.getValue()));  
 }  
 });  
  
 mediaPlayer.setOnReady(new Runnable() {  
 @Override  
 public void run() {  
 javafx.util.Duration total = media.getDuration();  
 progressBar.setMax(total.toSeconds());  
 }  
 });  
 HistoryDecorator history = new saveIt(new addSingle()); // decorator  
 history.add("+1");  
 mediaPlayer.play();  
 }  
 }  
  
 @Override  
 public void initialize(URL url, ResourceBundle rb) {  
   
 }  
  
 public void pauseVideo(ActionEvent event){  
  
 mediaPlayer.pause();  
 }  
  
 public void stopVideo(ActionEvent event){  
  
 mediaPlayer.stop();  
 }  
  
 public void playVideo(ActionEvent event){  
 Actions play = new Actions.Builder()  
 .setName("Play")  
 .setType("Click Button")  
 .setDescription("Starts the video").build();  
 actiuni.add(play);  
 /\*for ( Actions actiune : actiuni) {  
 System.out.println(actiune.getName());  
 System.out.println(actiune.getDescription());  
 System.out.println(actiune.getType());  
 }\*/  
  
 mediaPlayer.play();  
 mediaPlayer.setRate(1);  
 }  
  
 public void skip5(ActionEvent event){  
 mediaPlayer.seek(mediaPlayer.getCurrentTime().add(javafx.util.Duration.seconds(5)));  
 }  
  
 public void furtherSpeedUpVideo(ActionEvent event){  
 mediaPlayer.setRate(2);  
 }  
  
 public void back5(ActionEvent event){  
 mediaPlayer.seek(mediaPlayer.getCurrentTime().add(javafx.util.Duration.seconds(-5)));  
 }  
  
 public void furtherSlowDownVideo(ActionEvent event){  
 mediaPlayer.setRate(0.5);  
  
 }  
  
  
  
}

Codul sursa al fisierului MediaPlayer.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Slider?>  
<?import javafx.scene.layout.BorderPane?>  
<?import javafx.scene.layout.HBox?>  
<?import javafx.scene.layout.StackPane?>  
<?import javafx.scene.layout.VBox?>  
<?import javafx.scene.media.MediaView?>  
  
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="mediaplayer.MediaPlayerController">  
 <center>  
 <StackPane fx:id="pane" prefHeight="150.0" prefWidth="200.0" BorderPane.alignment="CENTER">  
 <children>  
 <MediaView fx:id="mediaView" fitHeight="200.0" fitWidth="200.0" />  
 <VBox alignment="BOTTOM\_CENTER" maxHeight="40.0" prefHeight="50.0" StackPane.alignment="BOTTOM\_CENTER">  
 <children>  
 <Slider fx:id="progressBar" />  
 <HBox alignment="BOTTOM\_CENTER" maxHeight="40.0" prefHeight="30.0">  
 <children>  
 <Button fx:id="openFile" mnemonicParsing="false" onAction="#OpenFileMethod" text="OpenFile">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <Button mnemonicParsing="false" onAction="#playVideo" text="Play">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <Button mnemonicParsing="false" onAction="#pauseVideo" text="Pause">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <Button mnemonicParsing="false" onAction="#stopVideo" text="Stop">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <!--<Button mnemonicParsing="false" onAction="#furtherSlowDownVideo" text="&lt;&lt;&lt;">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>-->  
 <Button mnemonicParsing="false" onAction="#back5" text="-5s">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <Button mnemonicParsing="false" onAction="#skip5" text="+5s">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>  
 <!-- <Button mnemonicParsing="false" onAction="#furtherSpeedUpVideo" text="&gt;&gt;&gt;">  
 <HBox.margin>  
 <Insets bottom="7.0" right="7.0" />  
 </HBox.margin>  
 </Button>-->  
 <Slider fx:id="volumeSlider">  
 <HBox.margin>  
 <Insets bottom="10.0" />  
 </HBox.margin>  
 </Slider>  
 </children>  
 </HBox>  
 </children>  
 <StackPane.margin>  
 <Insets />  
 </StackPane.margin>  
 </VBox>  
 </children>  
 </StackPane>  
 </center>  
</BorderPane>