Shaikh Saud
10LF333

MIP – Proiect Laborator

Modul în care voi aborda subiectele de barem nu va fi in ordinea dată, ci în ordinea aparițiilor și importanței acestora în implementarea mini-sistemului conceput, deci sper că nu vor fi probleme întrucât este aclași material acoperit.

1) Interfețe. (Lab. 6)

Pentru fiecare din clasele de bază am scris câte o interfață pentru a-mi stabili clar ce funcționalități trebuie să acopere aplicația și nu numai, ce metode trebuie testate ulterior. Am ajuns astfel la 3 interfețe: IController – pentru scheletele logicii funționării aplicației, IView – pentru a mă orienta ce funționalități îi sunt oferite utilizatorului din interfața propriu-zisă și IRecord – pentru definriea minimnală a modelelor de date care vor fi folosite.

```
public interface IRecord {  35 usages  3 implementations  saudz12
    enum ContributionType{  39 usages  saudz12
        Producer,  11 usages
        Performer,  12 usages
        Writer,  11 usages
        Composer,  9 usages
    }

    boolean SetPrice(double price);  6 usages  1 implementation  saudz12
    double GetPrice();  3 usages  1 implementation  saudz12

    void SetName(String length);  4 usages  1 implementation  saudz12
    String GetName();  21 usages  1 implementation  saudz12

    public Triplet<Integer, Integer, Integer> GetLength();  19 usages  1 implementation  saudz12
    public boolean SetLength(int hours, int minutes, int seconds);  7 usages  1 implementation  saudz12
    public String LengthToString();  4 usages  1 implementation  saudz12

    boolean SetReleaseDate(String date);  9 usages  1 implementation  saudz12
    String GetReleaseDate();  4 usages  1 implementation  saudz12

    void SetGenres(Set<String> genres);  2 usages  1 implementation  saudz12
    Set<String> GetGenres();  12 usages  1 implementation  saudz12

    public Map<String, Object> MapObject();  4 usages  3 implementations  saudz12

    boolean AddArtist(String name, ContributionType contribution);  19 usages  2 implementations  saudz12

    boolean SetArtists(Set<String> artists, ContributionType contribution);  5 usages  2 implementations
    boolean RemoveContribution(String name, ContributionType contribution);  7 usages  2 implementations
    boolean RemoveArtist(String name);  3 usages  2 implementations  saudz12

    boolean AddGenre(String genre);  5 usages  1 implementation  saudz12
        boolean RemoveGenre(String genre);  2 usages  1 implementation  saudz12

}
```

2) Clase Abstracte, Moșteniri. (Lab. 5)

Scriind modelele ce implementau IRecord constatasem prezența unor metode si variabile comune, precum ‘name’, ‘length’, ‘release_date’, getteri și setteri pentru acestea și metode de

convertire a datelor într-un Map și într-un String pentru uz ulterior în cadrul aplicației. Toate acestea sunt abordate de clasa abstractă RecordBase. Acuma, clasele care implementau IRecord extind RecordBase, iar RecordBase implementează parțial interfața cu metodele comune între clasele finale moștenite.

```
public abstract class RecordBase implements IRecord {  8 usages  2 inheritors  saudz12
    private String name;  5 usages
    private String releaseDate;  5 usages
    private Triplet<Integer, Integer, Integer> length;  5 usages
    private Set<String> genres;  7 usages
    private double price;  5 usages

    public RecordBase(String name){  2 usages  saudz12
        if(name.isEmpty())
            throw new RuntimeException("Can't have empty album name");
        this.name = name;
        this.releaseDate = "";
        this.length = new Triplet<>(0, 0, 0);
        this.genres = new HashSet<String>();
        this.price = 0;
    }

    public RecordBase(String name, Triplet<Integer,Integer,Integer> length){  1 usage  saudz12
        if(name.isEmpty())
            throw new RuntimeException("Can't have empty album name");
        this.name = name;
        this.releaseDate = "";
        this.length = length;
        this.genres = new HashSet<String>();
        this.price = 0;
    }

    public RecordBase(String name, Triplet<Integer,Integer,Integer> length, double price){  1 usage  saudz12
        if(name.isEmpty())
            throw new RuntimeException("Can't have empty album name");
        this.name = name;
        this.releaseDate = "";
        this.length = length;
        this.genres = new HashSet<String>();
        this.price = price;
    }

    @Override  4 usages  2 overrides  saudz12
    public Map<String, Object> MapObject(){
        Map<String, Object> jsonMap = new HashMap<>();
        jsonMap.put("name", this.GetName());
        jsonMap.put("release_date", this.GetReleaseDate());
        jsonMap.put("length", this.LengthToString());
        jsonMap.put("price", this.GetPrice());
        jsonMap.put("genres", this.GetGenres().toArray());


        return jsonMap;
    }
```

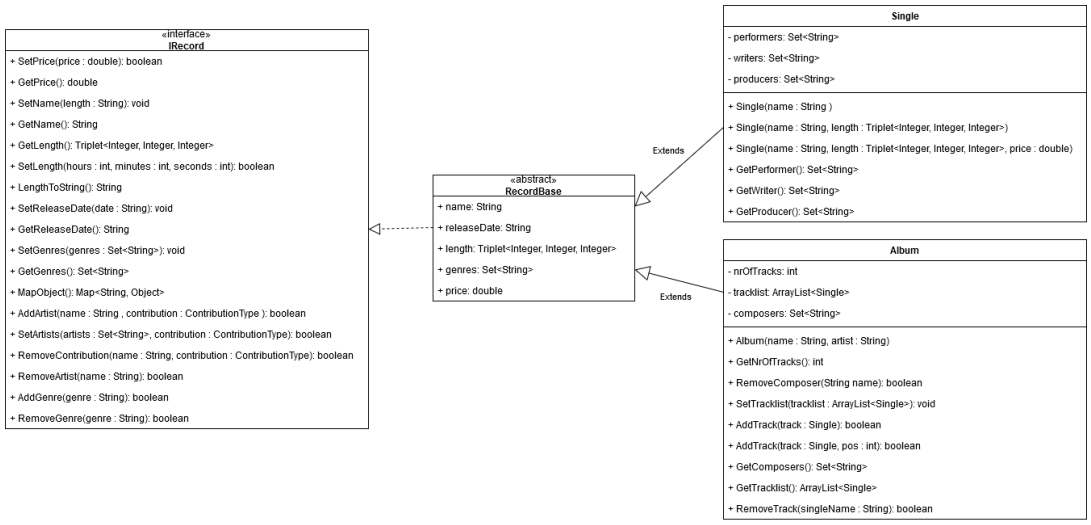
3) Clasele în limbajul java. (Lab. 4)

Ca să contrinuăm cu subpunctul anterior, voi nota despre clasele Single și Album. Ambele mostenesc și extind RecordBase dar în contexte diferite: Single este gândit ca un descriptor al unui obiect unic, cât timp Album al unei colecții. Drept metode ambele folosesc aceiași getteri și setteri implementați în clasa abstractă, metode suprascrise pentru maparea membrilor și metode unice prezente doar în clasa respectivă (în cazul Albumului AddTrack) pentru definirea funționalităților acestuia.

Shaikh Saud
10LF333

```java
public class Album extends RecordBase{  19 usages  saudz12
    private int nrOfTracks;  7 usages
    private ArrayList<Single> tracklist;  12 usages
    private Set<String> composers;  9 usages

    public Album(String name, String artist){  11 usages  saudz12
        super(name);
        if(artist.isEmpty())
            throw new RuntimeException("Can't have empty artist name");
        this.tracklist = new ArrayList<Single>();
        this.composers = new HashSet<String>();
        this.composers.add(artist);
    }

    @Override  4 usages  saudz12
    public Map<String, Object> MapObject(){
        Map<String, Object> jsonMap = super.MapObject();
        jsonMap.put("nr_of_tracks", this.nrOfTracks);

        Map<String, Object> trackMap = new HashMap<>();

        for (Single s  : tracklist){
            trackMap.put(s.GetName(), s.MapObject());
        }

        jsonMap.put("tracklist", trackMap);
        jsonMap.put("composers", composers.toArray());

        return jsonMap;
    }
```

După implementarea completă a celor 3 aspecte, ajugem la ierarhia (im1), unde regăsim relații de **realizare** (implementare) marcate prin săgeată punctată si **generalizare** marcate prin săgeata normală.



(IM1)

Alte clase care apar sunt Menu ce moștenește IController și View ce moștenește interfața IView.

Shaikh Saud
10LF333

4) Metode, citiri, afișări, structuri logice și tipuri de date. (Lab. 1+2)

```
@Override  2 usages  saudz12
public void EditTrack(){
    ViewTracklist();
    int nr = menu.activeAlbum.GetNrOfTracks();
    if (nr == 0)
    {
        System.out.println("No tracks available: ");
        return;
    }
    System.out.println("Select a track to edit: ");
    String name = in.nextLine();
    if(menu.SelectTrack(name)){
        System.out.println("Track selected successfully");
    }
    else{
        System.out.println("Track couldn't be selected..");
        return;
    }
    while (true){
        menu.ViewTrackInfo(name);
        PrintTrackOptions();
        int option = in.nextInt();
        switch (option){
            case 0:
                return;
            case 1:{
                Flush();
                ChangeTrackName();
                break;
            }
            case 2:{
                Flush();
                ChangeTrackDuration();
                break;
            }
            case 3:{
                Flush();
                ChangeTrackReleaseDate();
                break;
            }
            case 4:{
```

În metoda EditTrack prezentă în clasa Menu observăm toate obiectivele subpunctului prezente:

- Afisarea în consolă a mesajelor de eroare și ajutătoare – System.out.println(…);
- Citirea de la tastatură din consolă a unui String și număr întreg. – in.next() și in.nextInt() (** in este un membru de tip Scanner inițializat în constructorul clasei cu delimitator '\n' pentru a se putea citi linii întregi în cazul String-urilor - *in = new Scanner(System.in).useDelimiter("\\n");)*
- Prezența instrucțiunilor de tip IF pentru decizii legate de continuarea sau nu a parcurgerii metodei, SWITCH pentru alegerea acțiunii pe care o va lua programul în continuare și WHILE pentru repetiția acestora după cerințele utilizatorului.

5) Colecții și Persistența datelor în java (Lab. 3+8)

Sistemul are la bază salvarea fiecărui obiect creat și existent în fișiere .json separate a căror date vor fi încărcate și parsate la începutul programului sau când un album este specific cerut să fie încărcat din memorie de către utilizator. .json handling-ul se face prin pachetul jackson.

În constructorul clasei Menu ne încărcăm din json toate albumele reținute local:

```
public Menu(){  1 usage  saudz12
    albums = new ArrayList<>();
    JSONObject jsonObject;
    try {
        File myObj = new File( pathname: Menu.pathname + "source.json");
        if (!myObj.createNewFile()) {
            ObjectMapper objectMapper = new ObjectMapper();
            Map<String, List<String>> jsonMap = objectMapper.readValue(
                    new File( pathname: Menu.pathname + "source.json"),
                    Map.class
            );

            // Extract the "albums" list from the Map
            List<String> albumsList = jsonMap.get("albums");

            // Convert the list to an ArrayList
            this.albums = new ArrayList<>(albumsList);

        }
        else {
            System.out.println("File created!");
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Shaikh Saud
10LF333

In cadrul clasei RecordBase și celor care o moștenesc regăsim funția MapObject care ne pregatește datele unui obiect pentru a fi parsate într-un .json ulterior folosind object mapper-ul din jackson. În procesul convertirii este folosit un Map având cheia un String semnificativ cheii cu care va apărea membrul parsat în fisier și Object valoare, tipul acestuia variând(String, Array, int alt Map etc.). Ne folosim de metoda **.put** a clasei Map pentru a adăuga perechiile de tip **<key, value>** mapei.

```
public Map<String, Object> MapObject(){
    Map<String, Object> jsonMap = super.MapObject();
    jsonMap.put("nr_of_tracks", this.nrOfTracks);

    Map<String, Object> trackMap = new HashMap<>();

    for (Single s  : tracklist){
        trackMap.put(s.GetName(), s.MapObject());
    }

    jsonMap.put("tracklist", trackMap);
    jsonMap.put("composers", composers.toArray());

    return jsonMap;
}
```

6) Unit Testing (Lab. 7)

În număr de 9 teste au fost întocmite pentru a ne asigura că obiectele se construiesc bine și că metodele acestora produc rezultate așteptate.

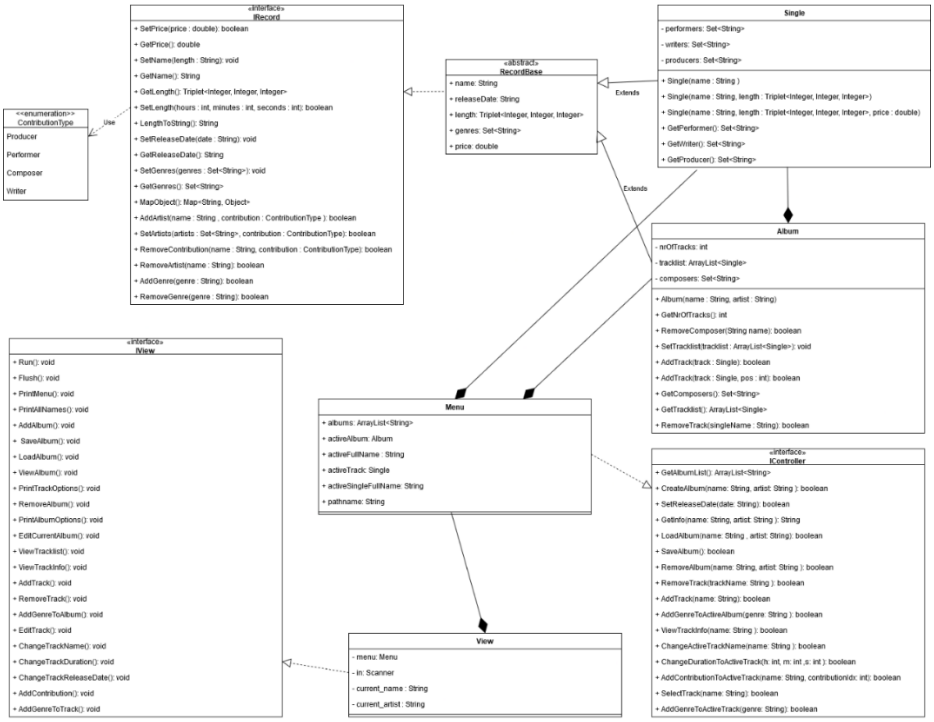
```
import org.javatuples.Triplet;
import org.junit.jupiter.api.Test;

import java.util.Set;
import java.util.HashSet;

import static org.junit.jupiter.api.Assertions.*;

class UnitTests {  ± saudz12

    ///RECORD BASE
    @Test void RecordNameTests(){...}

    @Test void RecordPriceTest(){...}

    @Test void RecordLengthTest(){  ± saudz12
        Single s = new Single( name: "test");
        assertTrue(s.SetLength( hours: 1,  minutes: 3,  seconds: 45));
        assertTrue( condition: s.GetLength().getValue0() == 1 && s.GetLength().getValue1() == 3 && s.GetLength().getValue2() == 45);
        assertFalse(s.SetLength( hours: 9999,  minutes: 2,  seconds: -1));

        Album a = new Album( name: "test2",  artist: "test3");
        Triplet<Integer, Integer, Integer> duration = new Triplet<>(0, 0, 0);
        a.AddTrack(s);
        a.AddTrack(s);
        a.AddTrack(s);
        for(var track: a.GetTracklist()){
            duration.setAt0(duration.getValue0() + track.GetLength().getValue0());
            duration.setAt1(duration.getValue0() + track.GetLength().getValue1());
            duration.setAt2(duration.getValue0() + track.GetLength().getValue2());
        }
        assertTrue( condition: duration.getValue0() == a.GetLength().getValue0() && duration.getValue1() == a.GetLength().getValue1() && duration.getValue2() == a.GetLength().getValue2());

    }

    @Test void RecordReleaseTest(){...}
```

(Test Module sample)

Shaikh Saud
10LF333

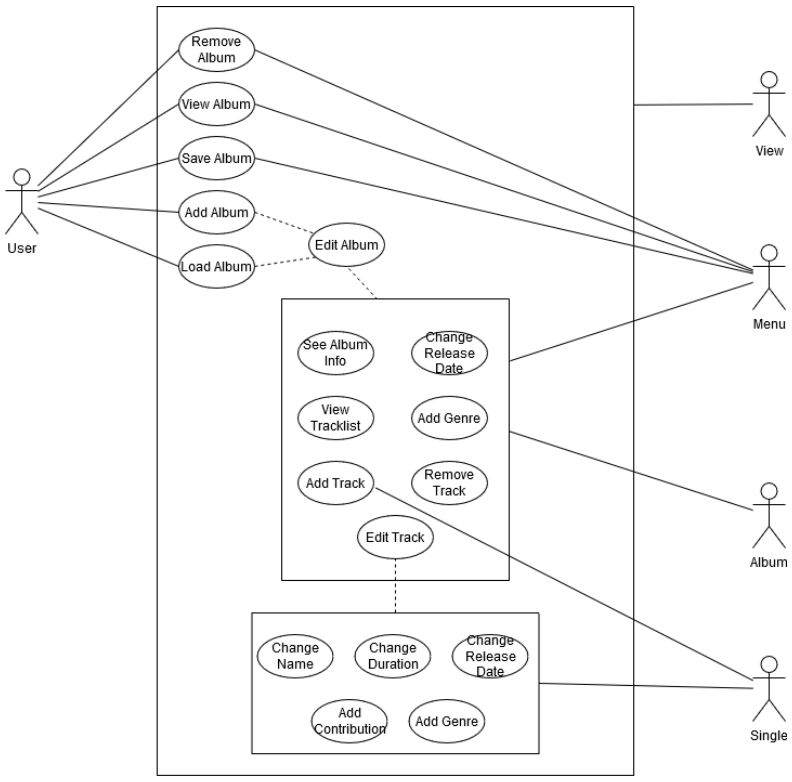
7) UML Final al aplicației (Lab. 9)

1. Class diagram



2. Use Case diagram

3.



Shaikh Saud
10LF333

3. Sequence diagram