Team No:16
Team members:

- Chegu sai poorna chandu(2022201062)
- Nemalikanti V M Dheeraj (2022201022)
- Patha Sai Sumith (2022202004)
- Krishna Chaitanya Dama (2021202005)
- Prem sai kumar kanamarlapudi (2022201036)

Task1:

User Management:
->Classes used and their description

Album Class:
It has the variables related to an album like artistid, album name, location, created and updated date, and directory id. It has the corresponding getter and setter methods
It has a composition relation with the Artist and UserAlbum class. The album class cannot exist without the Artist class and the User Album class cannot exist without the Album class.

Artist Class:
It has variables like artist name, id and created and deleted date. It has the corresponding getter and setter methods as all the variables are private.

UserAlbum Class:
It has variables like userid, albumid, score, created date, and deleted date.It has the corresponding getter and setter methods as all the variables are private.

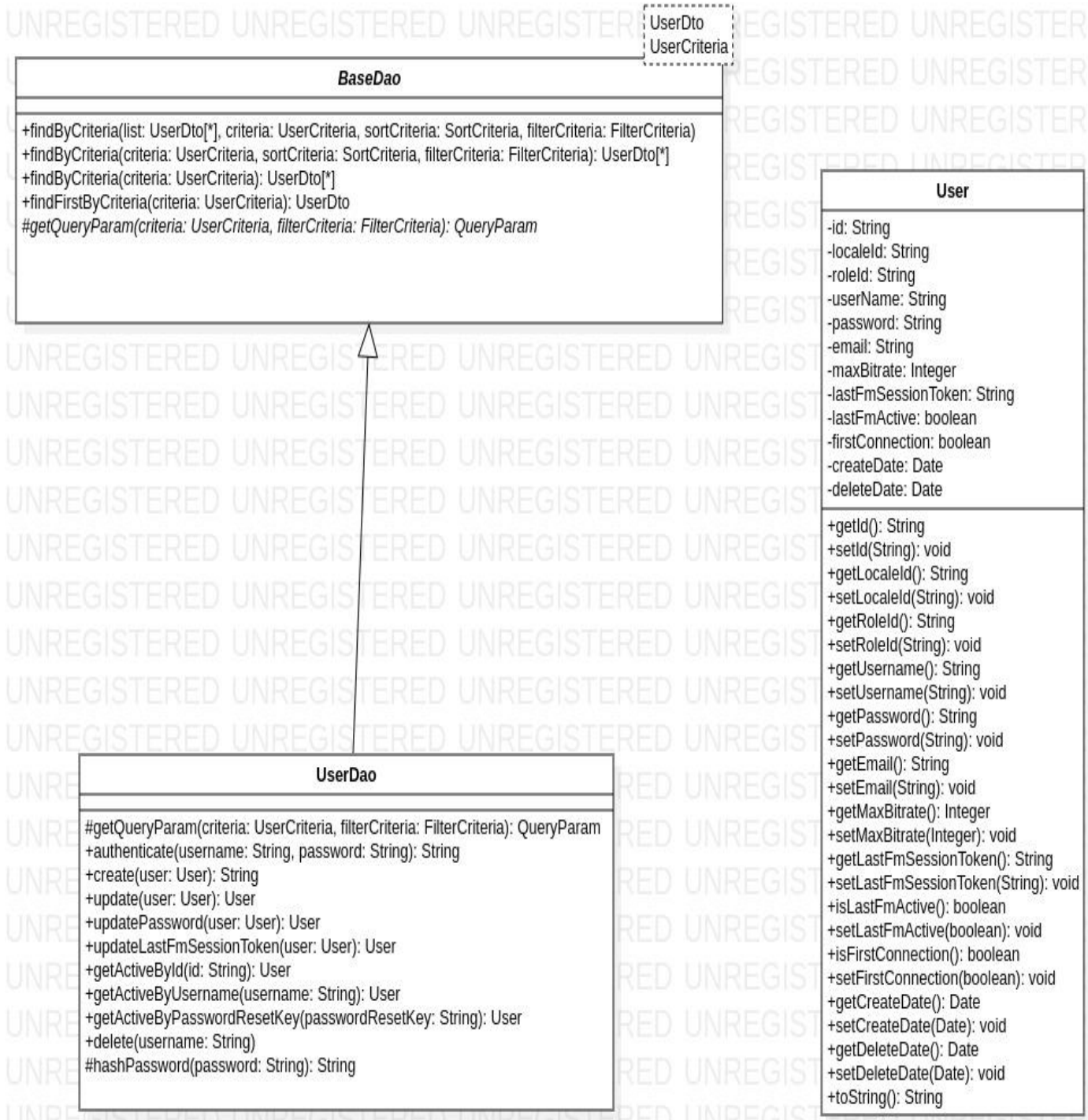
User Class:
It has variables like localeid, roleid, username, password, email, maxBitrate, lastFmSessionToken, lastFmActive, firstConnection, creation date, and deletedate.It has the corresponding getter and setter methods. It has composition relation with UserTrack, Playlist and UserAlbum class.

Playlist:
It has variables like playlistid, userid, and name. It has getter and setter methods. It has other methods like createPlayList(), updatePlayList(), and deletePlayList() to create, update and delete playlist respectively.

PlayListTrack:
It has variables like playlisttrack id, playlist id, trackId, and order. It has corresponding getter and setter methods and additionally createPlayListTrack(). It has association relation with Playlist class.

Class Track:
It has variables id, album id, artist id, filename, title, titleCorrected, year, genre, length, bitrate, order, vbr, format, createDate, and deleteDate.It has the corresponding getter and setter methods.

UserTrack Class:
It has variables like id, userid, trackid, playcount, like, createdate, and deletedate.It has the corresponding getter and setter methods.



If not clear visit:

um.jpg

Library Management :

BaseDao Class<ArtistDto, ArtistCriteria> : It has 3 findByCriteria() public methods overloaded by different Artist criterias, findFirstByCriteria() public method and getQueryParam() protected method.

ArtistDao Class : It has create(), update(), getActiveByName(), getActiveById(), delete(), deleteEmptyArtist() public methods, assembleResultList() private method and getQueryParam() protected method. It is inherited from the above BaseDao class.

BaseDao Class<PlaylistDto, PlaylistCriteria> : It has 3 findByCriteria() public methods overloaded by different Playlist criterias, findFirstByCriteria() public method and getQueryParam() protected method.

PlaylistDao Class : It has create(), update(), delete(), getDefaultPlaylistByUserId() public methods, assembleResultList() private method and getQueryParam() protected method. It is inherited from the above BaseDao class.

PlaylistTrackDao Class : It has create(), deletePlaylistById(), getPLaylistTrackNextOrder(), insertPlaylistTrack(), removePlaylistTrack() public methods.

BaseDao Class<AlbumDto, AlbumCriteria> : It has 3 findByCriteria() public methods overloaded by different Album criterias, findFirstByCriteria() public method and getQueryParam() protected method.

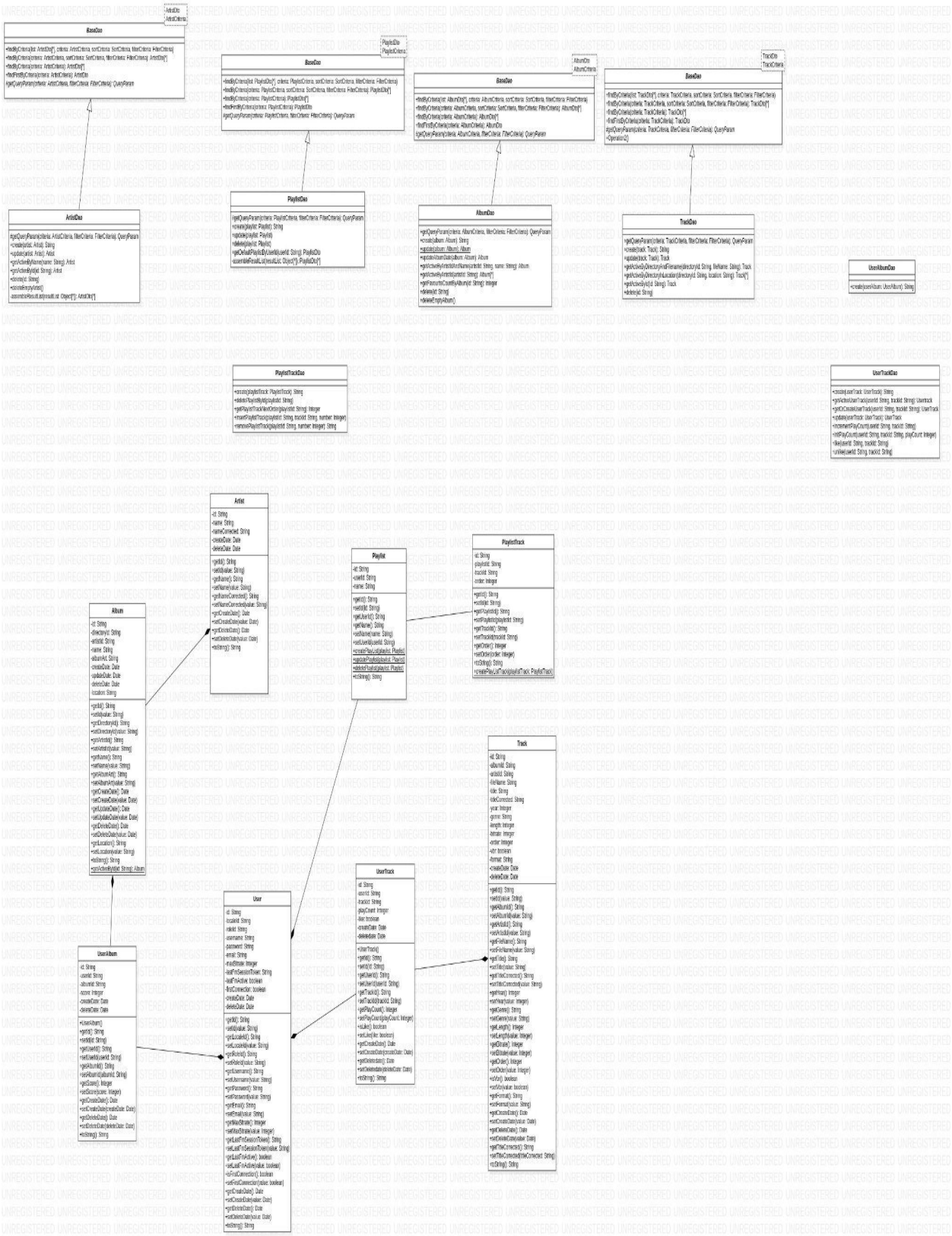
AlbumDao Class : It has create(), update(), updateAlbumDate(), getActiveByArtistId(), getActiveByArtistIdAndName(), getFavouriteCountByAlbum(), delete(), deleteEmptyAlbum(), getQueryParam() public methods. It is inherited from the above BaseDao class.

BaseDao Class<TrackDto, TrackCriteria> : It has 3 findByCriteria() public methods overloaded by different Track criterias, findFirstByCriteria() public method and getQueryParam() protected method.

TrackDao Class : It has create(), update(), getActiveByDirectoryAndFilename(), getActiveByDirectoryInLocation(), getActiveById(), delete(), getQueryParam() public methods. It is inherited from the above BaseDao class.

UserAlbumDao Class : It has a create() public method.

UserTrackDao Class : It has create(), getActiveUserTrack(), getOrCreateUserTrack(), update(), incrementPlayCount(), initPlayCount(), like(), unlike() public methods.


BaseDao
ArtistDto
ArtistCriteria
+findByCriteria(list: ArtistDto[*], criteria: ArtistCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria)
+findByCriteria(criteria: ArtistCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria): ArtistDto[*]
+findByCriteria(criteria: ArtistCriteria): ArtistDto[*]
+findFirstByCriteria(criteria: ArtistCriteria): ArtistDto
#getQueryParam(criteria: ArtistCriteria, filterCriteria: FilterCriteria): QueryParam
BaseDao
PlaylistDto
PlaylistCriteria
+findByCriteria(list: PlaylistDto[*], criteria: PlaylistCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria)
+findByCriteria(criteria: PlaylistCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria): PlaylistDto[*]
+findByCriteria(criteria: PlaylistCriteria): PlaylistDto[*]
+findFirstByCriteria(criteria: PlaylistCriteria): PlaylistDto
#getQueryParam(criteria: PlaylistCriteria, filterCriteria: FilterCriteria): QueryParam
BaseDao
AlbumDto
AlbumCriteria
+findByCriteria(list: AlbumDto[*], criteria: AlbumCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria)
+findByCriteria(criteria: AlbumCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria): AlbumDto[*]
+findByCriteria(criteria: AlbumCriteria): AlbumDto[*]
+findFirstByCriteria(criteria: AlbumCriteria): AlbumDto
#getQueryParam(criteria: AlbumCriteria, filterCriteria: FilterCriteria): QueryParam
BaseDao
TrackDto
TrackCriteria
+findByCriteria(list: TrackDto[*], criteria: TrackCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria)
+findByCriteria(criteria: TrackCriteria, sortCriteria: SortCriteria, filterCriteria: FilterCriteria): TrackDto[*]
+findByCriteria(criteria: TrackCriteria): TrackDto[*]
+findFirstByCriteria(criteria: TrackCriteria): TrackDto
#getQueryParam(criteria: TrackCriteria, filterCriteria: FilterCriteria): QueryParam
+Operator2[]
ArtistDao
#getQueryParam(criteria: ArtistCriteria, filterCriteria: FilterCriteria): QueryParam
+create(artist: Artist): String
+update(artist: Artist): Artist
+getOrCreateByName(name: String): Artist
+getActiveById(id: String): Artist
+delete(id: String)
+deleteEmptyArtist()
-assembleResult(rs:ps:result:List Object[*]): ArtistDto[*]
PlaylistDao
#getQueryParam(criteria: PlaylistCriteria, filterCriteria: FilterCriteria): QueryParam
+create(playlist: Playlist): String
+update(playlist: Playlist)
+delete(playlist: Playlist)
+getDefaultPlaylistByUserId(userId: String): PlaylistDto
-assembleResultList(resultList: Object[*]): PlaylistDto[*]
AlbumDao
+getQueryParam(criteria: AlbumCriteria, filterCriteria: FilterCriteria): QueryParam
+create(album: Album): String
+update(album: Album): Album
+updateAlbumDate(album: Album): Album
+getActiveByArtistIdAndName(artistId: String, name: String): Album
+getActiveByArtistId(artistId: String): Album[*]
+getPersonalTrackedByAlbum(id: String): Integer
+delete(id: String)
+deleteEmptyAlbum()
TrackDao
+getQueryParam(criteria: TrackCriteria, filterCriteria: FilterCriteria): QueryParam
+create(track: Track): String
+update(track: Track): Track
+getActiveByDirectoryAndFilename(directoryId: String, fileName: String): Track
+getActiveByDirectoryAndLocation(directoryId: String, location String): Track[*]
+getActiveById(id: String): Track
+delete(id: String)
UserAlbumDao
+create(userAlbum: UserAlbum): String
PlaylistTrackDao
+create(playlistTrack: PlaylistTrack): String
+deleteByPlaylistId(playlistId: String)
+getPlaylistTrackNextOrder(playlistId: String): Integer
+insertPlaylistTrack(playlistId: String, trackId: String, number: Integer)
+removePlaylistTrack(playlistId: String, number: Integer): String
UserTrackDao
+create(userTrack: UserTrack): String
+getActiveUserTrack(userId: String, trackId: String): UserTrack
+getOrCreateUserTrack(userId: String, trackId: String): UserTrack
+update(userTrack: UserTrack): UserTrack
+incrementPlayCount(userId: String, trackId: String)
+setPlayCount(userId: String, trackId: String, playCount: Integer)
+like(userId: String, trackId: String)
+unlike(userId: String, trackId: String)
Artist
-id: String
-name: String
-nameCorrected: String
-createDate: Date
-deleteDate: Date
+getId(): String
+setId(value: String)
+getName(): String
+setName(value: String)
+getNameCorrected(): String
+setNameCorrected(value: String)
+getCreateDate(): Date
+setCreateDate(value: Date)
+getDeleteDate(): Date
+setDeleteDate(value: Date)
+toString(): String
Playlist
-id: String
-userId: String
-name: String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(userId: String)
+getName(): String
+setName(name: String)
+setUserId(userId: String)
+createNewPlaylist(playlistId, PlaylistDto)
+updateModifiedDate(playlistId, Playlist)
+updateFinalId(playlistId, Playlist)
+toString(): String
PlaylistTrack
-id: String
-playlistId: String
-trackId: String
-order: Integer
+getId(): String
+setId(id: String)
+getPlaylistId(): String
+setPlaylistId(playlistId: String)
+getTrackId(): String
+setTrackId(trackId: String)
+getOrder(): Integer
+setOrder(order: Integer)
+toString(): String
+createNewListTrack(playlistTrack: PlaylistTrack)
Album
-id: String
-directoryId: String
-artistId: String
-name: String
-albumArt: String
-createDate: Date
-updateDate: Date
-deleteDate: Date
-location: String
+getId(): String
+setId(value: String)
+getDirectoryId(): String
+setDirectoryId(value: String)
+getArtistId(): String
+setArtistId(value: String)
+getName(): String
+setName(value: String)
+getAlbumArt(): String
+setAlbumArt(value: String)
+getCreateDate(): Date
+setCreateDate(value: Date)
+getLastUpdate(): Date
+setUpdateDate(value: Date)
+getDeleteDate(): Date
+setDeleteDate(value: Date)
+getLocation(): String
+setLocation(value: String)
+toString(): String
+getOrCreateByName(name: String): Album
Track
-id: String
-albumId: String
-artistId: String
-fileName: String
-title: String
-titleCorrected: String
-year: Integer
-genre: String
-length: Integer
-bitrate: Integer
-order: Integer
-chr: boolean
-format: String
-createDate: Date
-deleteDate: Date
+getId(): String
+setId(value: String)
+getAlbumId(): String
+setAlbumId(value: String)
+getArtistId(): String
+setArtistId(value: String)
+getFileName(): String
+setFileName(value: String)
+getTitle(): String
+setTitle(value: String)
+getTitleCorrected(): String
+setTitleCorrected(value: String)
+getYear(): Integer
+setYear(value: Integer)
+getGenre(): String
+setGenre(value: String)
+getLength(): Integer
+setLength(value: Integer)
+getBitrate(): Integer
+setBitrate(value: Integer)
+getOrder(): Integer
+setOrder(value: Integer)
+isChr(): boolean
+setChr(value: boolean)
+getFormat(): String
+setFormat(value: String)
+getCreateDate(): Date
+setCreateDate(value: Date)
+getDeleteDate(): Date
+setDeleteDate(value: Date)
+getTitleCorrected(): String
+setTitleCorrected(titleCorrected: String)
+toString(): String
User
-id: String
-locale: String
-nickId: String
-username: String
-password: String
-email: String
-maxBitrate: Integer
-lastFmSessionToken: String
-lastFmActive: boolean
-firstConnection: boolean
-createDate: Date
-deleteDate: Date
+getId(): String
+setId(value: String)
+getLocaleId(): String
+setLocaleId(value: String)
+getNickId(): String
+setNickId(value: String)
+getUsername(): String
+setUsername(value: String)
+getPassword(): String
+setPassword(value: String)
+getEmail(): String
+setEmail(value: String)
+getMaxBitrate(): Integer
+setMaxBitrate(value: Integer)
+getLastFmSessionToken(): String
+setLastFmSessionToken(value: String)
+getLastFmActive(): boolean
+setLastFmActive(value: boolean)
+isFirstConnection(): boolean
+setFirstConnection(value: boolean)
+getCreateDate(): Date
+setCreateDate(value: Date)
+getDeleteDate(): Date
+setDeleteDate(value: Date)
+toString(): String
UserTrack
-id: String
-userId: String
-trackId: String
-playCount: Integer
-like: boolean
-createDate Date
-deleteDate Date
+UserTrack()
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(userId: String)
+getTrackId(): String
+setTrackId(trackId: String)
+getPlayCount(): Integer
+setPlayCount(playCount: Integer)
+isLike(): boolean
+setLike(like: boolean)
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+toString(): String
UserAlbum
-id: String
-userId: String
-albumId: String
-score: Integer
-createDate: Date
-deleteDate: Date
+UserAlbum()
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(userId: String)
+getAlbumId(): String
+setAlbum(albumId: String)
+getScore(): Integer
+setScore(score: Integer)
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+toString(): String

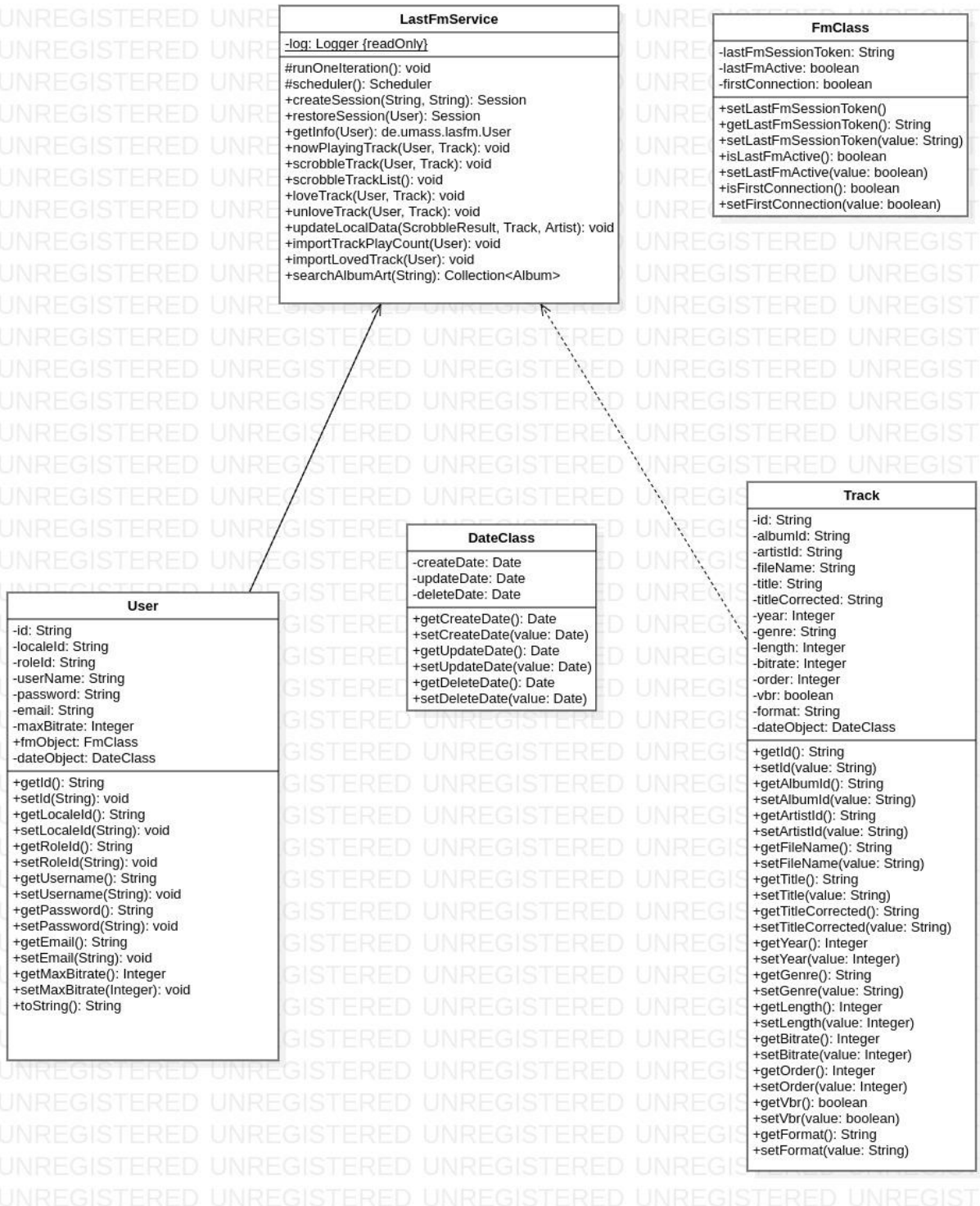
If not clear visit:
https://drive.google.com/file/d/1TOph4Lb7vSTIlGzY17_yQJg1VgLkxY8R/view?usp=sharing

Last.fm integration

Class LastFmService
It has a logger variable. It has protected methods runOneIteration() and scheduler(). It also has methods to createSession(), restoreSession(), getInfo(), nowPlayingTrack(), scrobbleTrack(), scrobbleTrackList(), loveTrack(), unloveTrack(), updateLocalTrack(), importTrackPlayCounter(), importLovedTrack(),searchAlbumArt().

Class Track:
It has variables id, album id, artist id, filename, title, titleCorrected, year, genre, length, bitrate, order, vbr, format, createDate, and deleteDate.It has the corresponding getter and setter methods.

User Class:
It has variables like localeid, roleid, username, password, email, maxBitrate, lastFmSessionToken, lastFmActive, firstConnection, creation date, and deletedate.It has the corresponding getter and setter methods. It has composition relation with UserTrack, Playlist and UserAlbum class.


LastFmService
-log: Logger {readOnly}
#runOneIteration(): void
#scheduler(): Scheduler
+createSession(String, String): Session
+restoreSession(User): Session
+getInfo(User): de.umass.lasfm.User
+nowPlayingTrack(User, Track): void
+scrobbleTrack(User, Track): void
+scrobbleTrackList(): void
+loveTrack(User, Track): void
+unloveTrack(User, Track): void
+updateLocalData(ScrobbleResult, Track, Artist): void
+importTrackPlayCount(User): void
+importLovedTrack(User): void
+searchAlbumArt(String): Collection<Album>
FmClass
-lastFmSessionToken: String
-lastFmActive: boolean
-firstConnection: boolean
+setLastFmSessionToken()
+getLastFmSessionToken(): String
+setLastFmSessionToken(value: String)
+isLastFmActive(): boolean
+setLastFmActive(value: boolean)
+isFirstConnection(): boolean
+setFirstConnection(value: boolean)
DateClass
-createDate: Date
-updateDate: Date
-deleteDate: Date
+getCreateDate(): Date
+setCreateDate(value: Date)
+getUpdateDate(): Date
+setUpdateDate(value: Date)
+getDeleteDate(): Date
+setDeleteDate(value: Date)
User
-id: String
-localeId: String
-roleId: String
-userName: String
-password: String
-email: String
-maxBitrate: Integer
+fmObject: FmClass
-dateObject: DateClass
+getId(): String
+setId(String): void
+getLocaleId(): String
+setLocaleId(String): void
+getRoleId(): String
+setRoleId(String): void
+getUsername(): String
+setUsername(String): void
+getPassword(): String
+setPassword(String): void
+getEmail(): String
+setEmail(String): void
+getMaxBitrate(): Integer
+setMaxBitrate(Integer): void
+toString(): String
Track
-id: String
-albumId: String
-artistId: String
-fileName: String
-title: String
-titleCorrected: String
-year: Integer
-genre: String
-length: Integer
-bitrate: Integer
-order: Integer
-vbr: boolean
-format: String
-dateObject: DateClass
+getId(): String
+setId(value: String)
+getAlbumId(): String
+setAlbumId(value: String)
+getArtistId(): String
+setArtistId(value: String)
+getFileName(): String
+setFileName(value: String)
+getTitle(): String
+setTitle(value: String)
+getTitleCorrected(): String
+setTitleCorrected(value: String)
+getYear(): Integer
+setYear(value: Integer)
+getGenre(): String
+setGenre(value: String)
+getLength(): Integer
+setLength(value: Integer)
+getBitrate(): Integer
+setBitrate(value: Integer)
+getOrder(): Integer
+setOrder(value: Integer)
+getVbr(): boolean
+setVbr(value: boolean)
+getFormat(): String
+setFormat(value: String)

If not clear visit:
https://drive.google.com/file/d/17FukJo-mupe95AlyXCrD_t6Y0lO4euaV/view?usp=sharing

Administration:

RolePrivilege
It has variables like id,roleId,privilegeId,createDate, and deleteDate.It has the corresponding getter and setter methods. It has a composition relationship with role and privilege class.

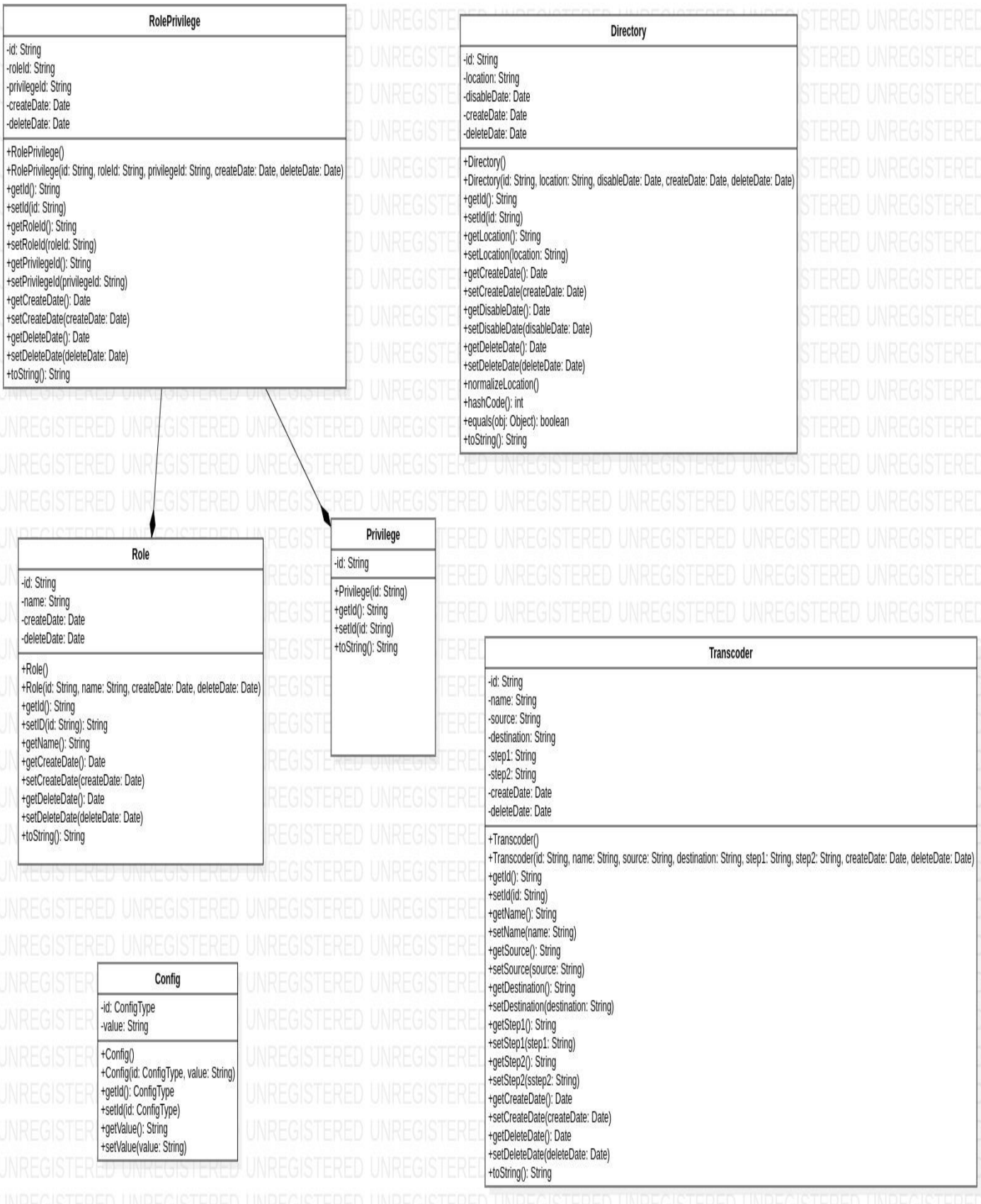
Directory:
It has variables like id, location,disableDate,createDate, and deleteDate.It has the corresponding getter and setter methods. Additionally, it has the normalizeLocation, hashCode, equals, and toString methods.

Transcoder:
It has variables like id, name, source, destination, step1, step2, createDate, and deleteDate. It has the corresponding getter and setter methods.

Role:
It has variables like name,createDate, and deleteDate.It has the corresponding getter and setter methods. It has a composition relationship with RolePrivilege class.

Privilege:
It has variables like stringId.It has getter and setter methods.It has composition relationship with RolePrivilege class.

Config:
It has variables like id,value.It has the corresponding getter and setter methods.


RolePrivilege
-id: String
-roleId: String
-privilegeId: String
-createDate: Date
-deleteDate: Date
+RolePrivilege()
+RolePrivilege(id: String, roleId: String, privilegeId: String, createDate: Date, deleteDate: Date)
+getId(): String
+setId(id: String)
+getRoleId(): String
+setRoleId(roleId: String)
+getPrivilegeId(): String
+setPrivilegeId(privilegeId: String)
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+toString(): String
Directory
-id: String
-location: String
-disableDate: Date
-createDate: Date
-deleteDate: Date
+Directory()
+Directory(id: String, location: String, disableDate: Date, createDate: Date, deleteDate: Date)
+getId(): String
+setId(id: String)
+getLocation(): String
+setLocation(location: String)
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDisableDate(): Date
+setDisableDate(disableDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+normalizeLocation()
+hashCode(): int
+equals(obj: Object): boolean
+toString(): String
Role
-id: String
-name: String
-createDate: Date
-deleteDate: Date
+Role()
+Role(id: String, name: String, createDate: Date, deleteDate: Date)
+getId(): String
+setID(id: String): String
+getName(): String
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+toString(): String
Privilege
-id: String
+Privilege(id: String)
+getId(): String
+setId(id: String)
+toString(): String
Transcoder
-id: String
-name: String
-source: String
-destination: String
-step1: String
-step2: String
-createDate: Date
-deleteDate: Date
+Transcoder()
+Transcoder(id: String, name: String, source: String, destination: String, step1: String, step2: String, createDate: Date, deleteDate: Date)
+getId(): String
+setId(id: String)
+getName(): String
+setName(name: String)
+getSource(): String
+setSource(source: String)
+getDestination(): String
+setDestination(destination: String)
+getStep1(): String
+setStep1(step1: String)
+getStep2(): String
+setStep2(sstep2: String)
+getCreateDate(): Date
+setCreateDate(createDate: Date)
+getDeleteDate(): Date
+setDeleteDate(deleteDate: Date)
+toString(): String
Config
-id: ConfigType
-value: String
+Config()
+Config(id: ConfigType, value: String)
+getId(): ConfigType
+setId(id: ConfigType)
+getValue(): String
+setValue(value: String)

If not clear visit:
https://drive.google.com/file/d/1vVCHazSnMTcSK217GdSFAqgEOc7cO15H/view?usp=sharing

Task 2
2a. Design and code smells
Path:
se-project-1-19-master/music-core/src/main/java/com/sismics/music/core/model/dbi

1. **Comments** code smell: Every class contains unnecessary comments. The variable and method names are self-explanatory.
2. Every class contains the method toString(). This is the **Duplicate Code** smell.
3. Also, this method toString() has **Message chaining** code smell.

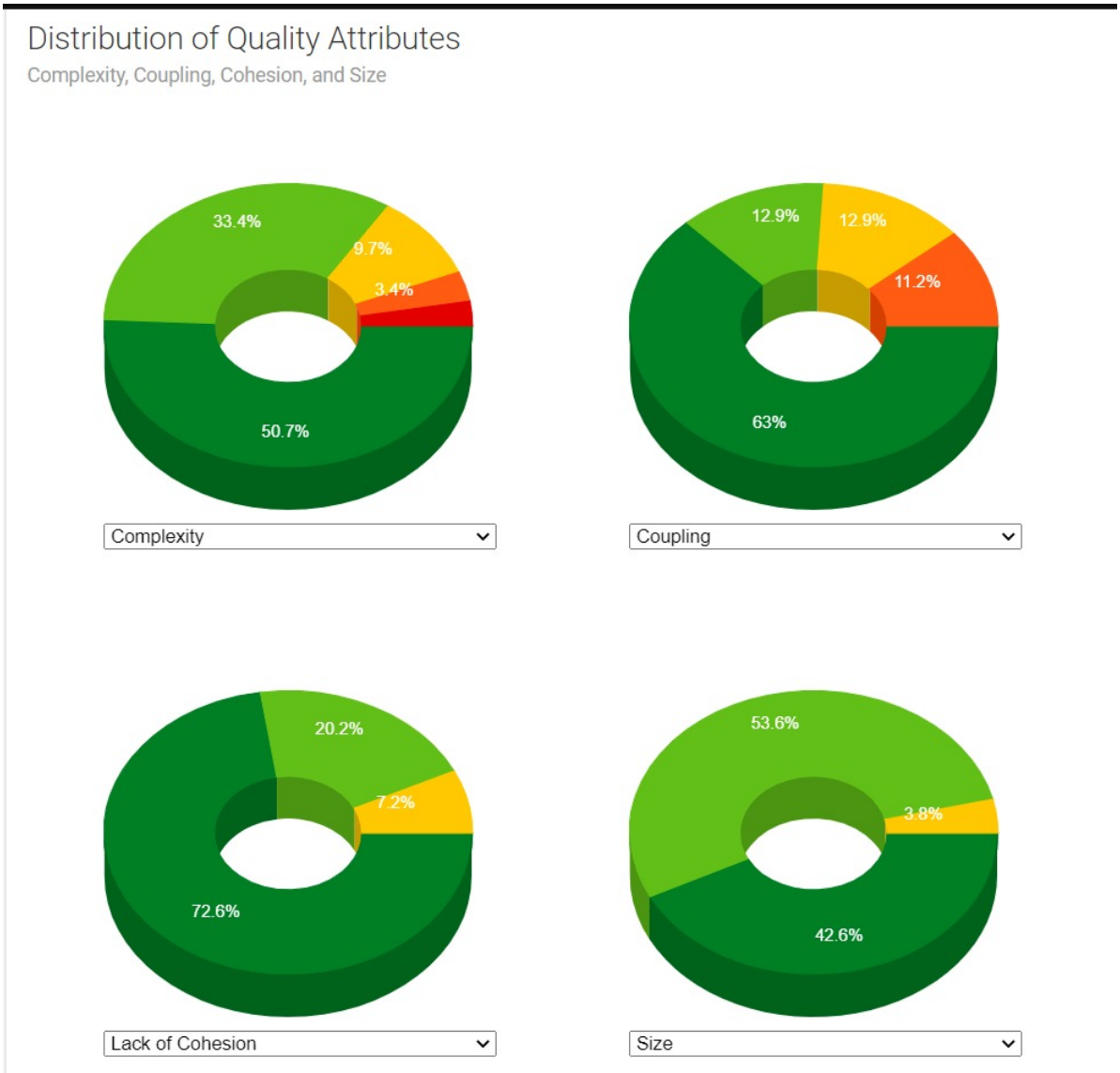
Path:
se-project-1-19-master/music-core/src/main/java/com/sismics/music/core/model/dbi

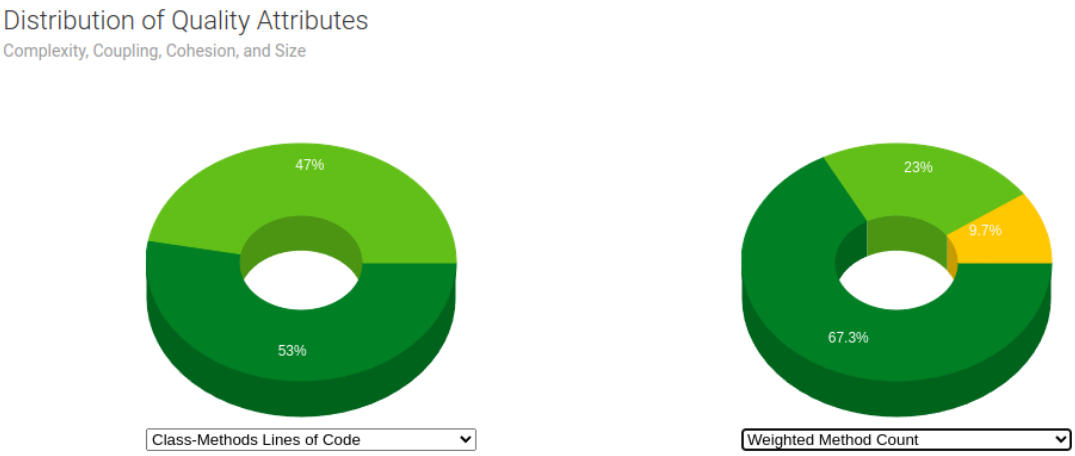
1. **Comments** code smell: Every class contains unnecessary comments. The variable and method names are self-explanatory.
2. Every class contains the method toString(). This is the **Duplicate Code** smell.
3. Also, this method toString() has **Message chaining** code smell.

Class	Code smell
Album	Primitive Obsession, Long Parameter List, Message Chaining
Artist	Primitive Obsession, Long Parameter List
AuthenticationToken	Primitive Obsession, Long Parameter List

Directory	Primitive Obsession, Long Parameter List
Playlist	Multiple chaining
PlaylistTrack	Primitive Obsession, Long Parameter List, message chaining
Role	Primitive Obsession, Long Parameter List

Task -2b





The graphs above depict the distribution of some of the metrics.
The original codebase had several weaknesses which were later improved in our design.

Metrics	**About**	**Strength of code**	**Weakness of code**
Complexity	It Implies being difficult to understand and describes the interactions between a number of entities. Higher levels of complexity in software increase the risk of unintentionally interfering with interactions and so increases the chance of introducing defects when making changes.	50.7% files have low complexity which means easy to understand and maintain and have less chance of errors or bugs	2% files have high complexity which means it becomes more complex, it can be more difficult to understand and maintain, as well as increase the likelihood of errors or bugs.
Coupling	Coupling between classes refers to the degree to which one class relies on another class. It is a	63% files have low coupling which means loose or minimal	3 classes have high coupling

	measure of how tightly two classes are connected or dependent on each other.	dependencies between each other	
Lack of Cohesion	refers to a situation where a module or a class in a software system has multiple responsibilities or functions that are not closely related to each other.	122 classes have low lack of cohesion tend to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability	3 classes with medium-high lack of cohesion which means low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand
size	Size is one of the oldest and most common forms of software measurement. Measured by the number of lines or methods in the code.	110 classes have a low size which means classes should not split up	2 classes with medium-high size means. A very high count might indicate that a class or method is trying to do too much work and should be split up. It might also indicate that the class might be hard to maintain.
Class-Methods Lines of Code	Total number of all nonempty, non-commented lines of methods inside a class.	117 classes with low-class methods lines of code	

Weighted method count	It is calculated by adding up the complexities of all the methods in a class, with each method being weighted according to its complexity.	126 classes with low weighted method count	3 classes with medium-high weighted method count

Path: se-project-1-19-master/music-core/src/main/java/com/sismics/music/core/service/albumart

Class	Code Smell
AlbumArtFileNameFilter	Return statement can be modified

Path: se-project-1-19-master/music-core/src/main/java/com/sismics/music/core/service/collection

Class	Code Smell
Collection Service	Remove unnecessary statements, long method, message chaining
Collection visitor	Message chaining
CollectionWatchService	Long method
ImportAudioService	Long parameter list, long method, message chaining

Design Smells:

Design smells are particular structures in the code that indicate violations of fundamental design principles and negatively impact the quality and maintainability of the software. They are symptoms of deeper design issues and can often lead to further problems down the line if left unaddressed. Design smells can be identified through code analysis techniques, and their presence often indicates that the code should be refactored to improve its overall design.

Missing Abstraction: A class suffers from the missing abstraction smell when resources and language elements are declared and used without encapsulating them in an abstraction.

Broken Modularization: Broken modularization refers to a design issue in which the modules or components of a software system are not well-defined, or the system does not follow the principles of modular design.

Imperative Abstraction: The "imperative abstraction" design smell is a code quality issue that occurs when code is written in an imperative style that uses low-level, step-by-step instructions to accomplish a task, rather than using a higher-level abstraction to express the intent of the code. This can make the code harder to understand, maintain, and modify because it is tightly coupled to the implementation details.

Type of design smell: Missing Abstraction
Class: User
New Class Name: FmClass
Removed attributes lastFmSessionToken,lastFmActive, and first connection from user class and created a new class named FmClass

Type of design smell: Missing Abstraction
Class: Album

New Class Name: DateClass
Removed attributes createDate,deleteDate,updateDate from album class and created new class named DateClass.

Type of design smell: Broken Modularization
Classes changed: removed classes Privilege and Role from administration and added it to RolePrivilige

Type of design smell: Imperative Abstraction
Path:se-project-1-16-master/music-core/src/main/java/com/sismics/music/core/event/async
DirectoryCreatedAsyncEvent and DirectoryDeletedAsyncEvent classes are merged as DirectoryCreatedOrDeletedAsyncEvent
LastFmUpdateLovedTrackAsyncEvent.java and LastFmUpdateTrackPlayCountAsyncEvent.java are merged as LastFmUpdateLovedOrPlayCounrTrackAsyncEvent
TrackLikedAsyncEvent and TrackUnlikedAsyncEvent are merged as TrackAsyncEvent
PlayStartedEvent and PlayCompletedEvent are merged PlayEvent.

Path:se-project-1-16-master/music-core/src/main/java/com/sismics/music/core/listener/async
DirectoryCreatedAsyncListener.java and DirectoryDeletedAsyncListener.java are merged as DirectoryCreatedOrDeletedAsyncListener
LastFmUpdateLovedTrackAsyncListener.java and LastFmUpdateTrackPlayCountAsyncListener.java are merged as LastFmUpdateLovedOrPlayCountTrackAsyncAsyncListener
PlayCompletedAsyncListener.java and PlayStartedAsyncListener.java are merged as PlayAsyncListener
TrackLikedAsyncListener.java and TrackUnlikedAsyncListener.java are merged as TrackAsyncListener.java

Task 3a Design Smells

1.)Missing Abstraction

It is located in com.sismics.music.core.model.dbi folder. We created a new class named FmClass having lastFmSessionToken, lastFmActive, and firstConnection variables. These three variables are related to each other, so we put them in a single class.

```
public class FmClass {
    3 usages
    private String lastFmSessionToken;
    3 usages
    private boolean lastFmActive;
    3 usages
    private boolean firstConnection;
    1 usage
    FmClass()
    {

    }
    1 usage
    FmClass( String lastFmSessionToken, boolean lastFmActive, boolean firstConnection)
    {
        this.lastFmSess
        this.lastFmActi
        this.firstConnection = firstConnection;
    }
    8 usages
    public String getLastFmSessionToken() { return lastFmSessionToken; }
    2 usages
    public void setLastFmSessionToken(String lastFmSessionToken) { this.lastFmSessionToken = lastFmSessionToken; }
    no usages
    public boolean isLastFmActive() { return lastFmActive; }
    no usages
    public void setLastFmActive(boolean lastFmActive) { this.lastFmActive = lastFmActive; }
    3 usages
    public boolean isFirstConnection() { return firstConnection; }
    1 usage
    public void setFirstConnection(boolean firstConnection) { this.firstConnection = firstConnection; }
}
```

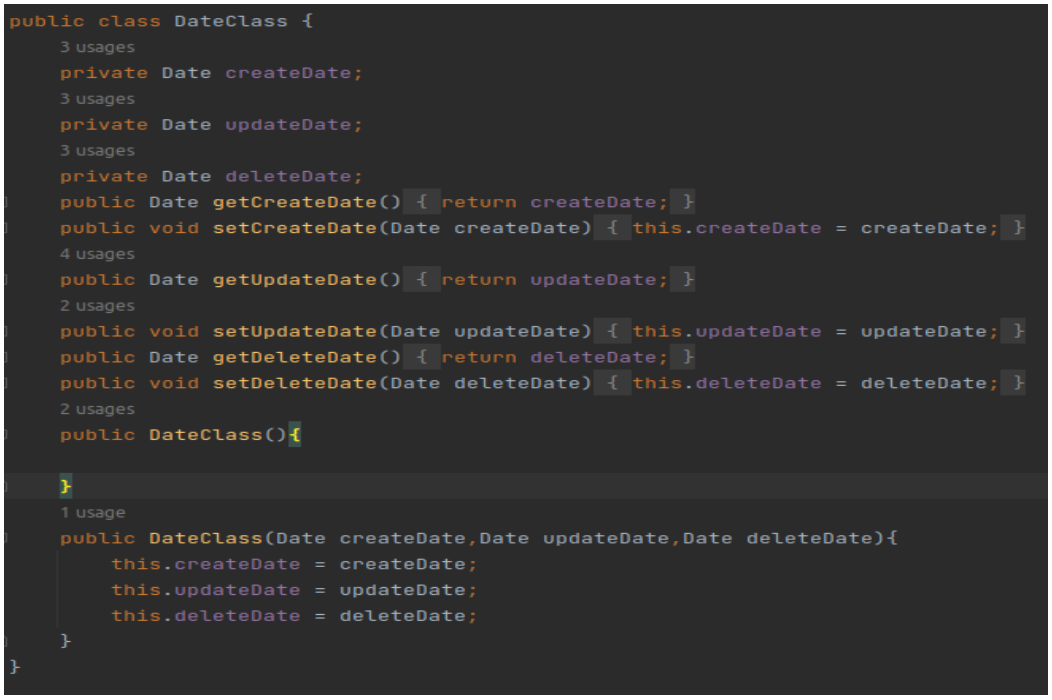
String lastFmSessionToken
music-core

We replace these fm-related variables with a single fmObject of type FmClass

```
16 usages
public FmClass fmObject;
/**
```

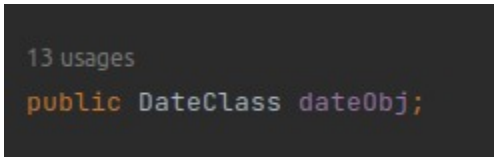
2.)Missing Abstraction

Similarly there are three variables createDate, deleteDate, and updateDate and they are related to each other. So we put them in a single class named DateClass.



```
public class DateClass {
    3 usages
    private Date createDate;
    3 usages
    private Date updateDate;
    3 usages
    private Date deleteDate;
    public Date getCreateDate() { return createDate; }
    public void setCreateDate(Date createDate) { this.createDate = createDate; }
    4 usages
    public Date getUpdateDate() { return updateDate; }
    2 usages
    public void setUpdateDate(Date updateDate) { this.updateDate = updateDate; }
    public Date getDeleteDate() { return deleteDate; }
    public void setDeleteDate(Date deleteDate) { this.deleteDate = deleteDate; }
    2 usages
    public DateClass(){

    }
    1 usage
    public DateClass(Date createDate,Date updateDate,Date deleteDate){
        this.createDate = createDate;
        this.updateDate = updateDate;
        this.deleteDate = deleteDate;
    }
}
```

So, we replace the three date-related variables with a single date object as follows.

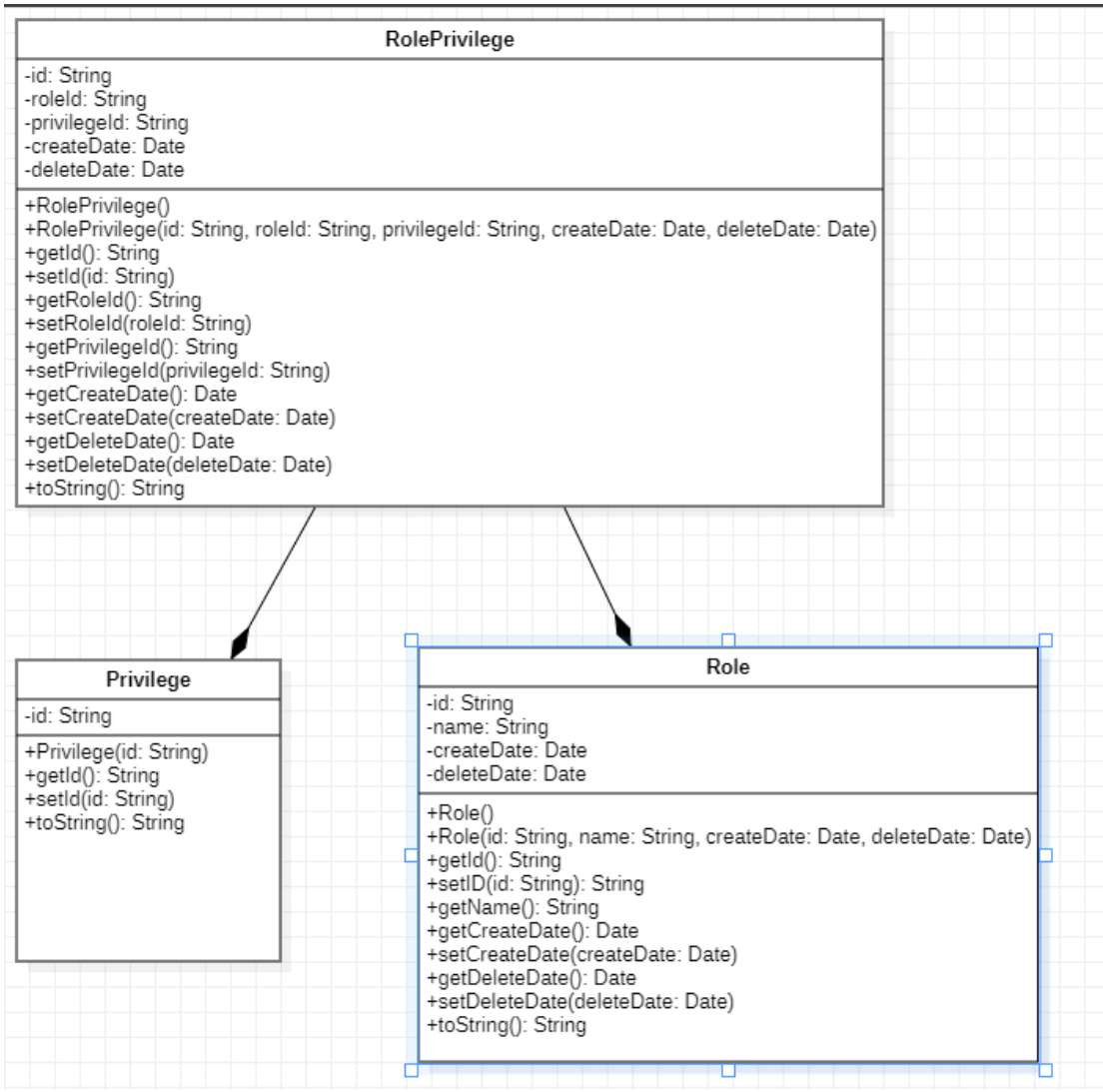


```
13 usages
public DateClass dateObj;
```

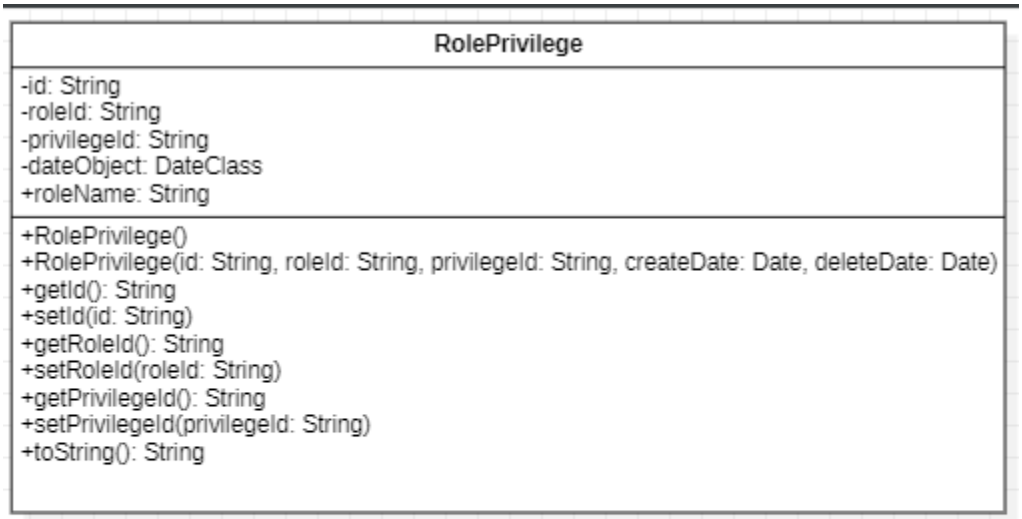
3.)Broken Modularization

We removed the Privilege and Role classes and added it to the RolePrivilege class.Below is the before and after changes

Before removing:



After removing



4.)Imperative Abstraction

```
final static FilenameFilter ALBUM_ART_FILENAME_FILTER = new AlbumArtFilenameFilter();
1 usage
public File scanDirectory(Path directory) {
    Map<Integer, File> fileMap = new TreeMap<>();
    for (File file : directory.toFile().listFiles(ALBUM_ART_FILENAME_FILTER)) { // XXX nio
        String name = file.getName().toLowerCase();
        if (name.startsWith("albumart.")) {
            fileMap.put( k: 0, file);
        } else if (name.startsWith("cover.")) {
            fileMap.put( k: 1, file);
        } else if (name.startsWith("front.")) {
            fileMap.put( k: 2, file);
        } else if (!fileMap.containsKey( o: 3)) {
            fileMap.put( k: 3, file);
        }
    }
    if (!fileMap.isEmpty()) {
        return fileMap.values().iterator().next();
    } else {
        return null;
    }
}
```

The above function is used only once. So this function is removed from AlbumArt class and implemented in collection service class.

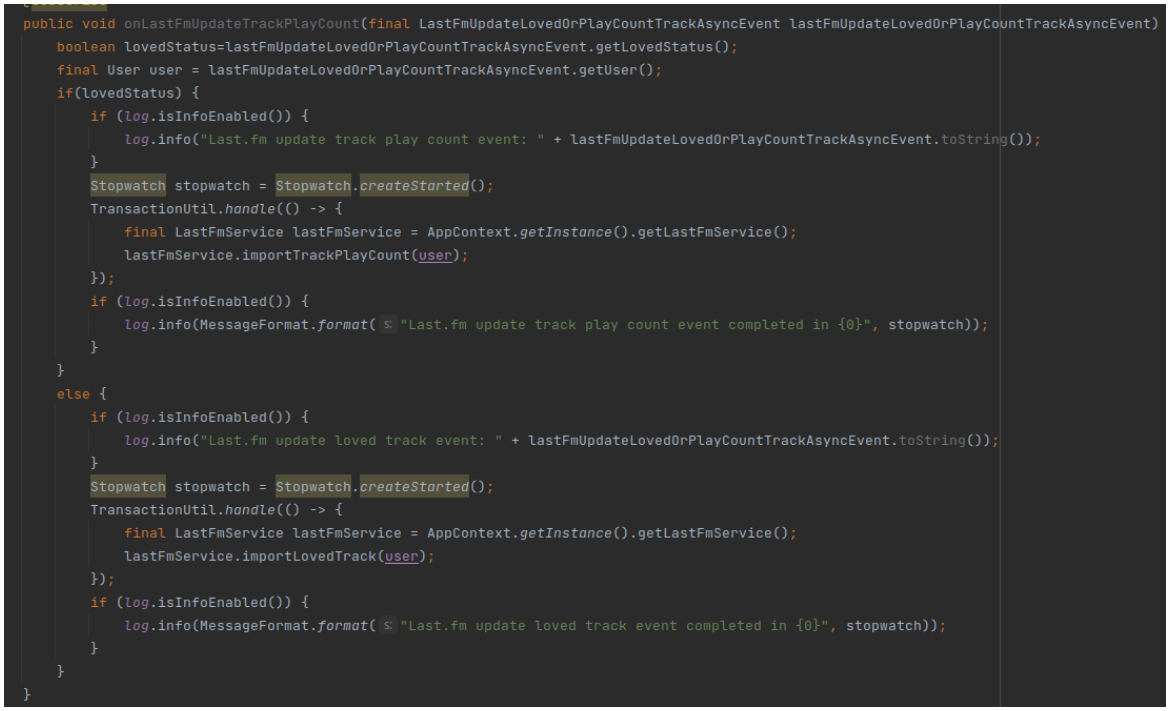
5.)Imperative Abstraction

In the Listener folder there are 2 classes named TrackUnlikedAsyncListener, and TrackLikedAsyncListener which have almost the same functionality except for liked and unliked. So we created a boolean variable named liked. The below code snippet clearly illustrates the above method.

```
boolean liked=trackAsyncEvent.getLiked();
if(liked)
{
    if (log.isInfoEnabled()) {
        log.info("Track liked event: " + trackAsyncEvent.toString());
    }
}
else
{
    if (log.isInfoEnabled()) {
        log.info("Track unliked event: " + trackAsyncEvent.toString());
    }
}

Stopwatch stopwatch = Stopwatch.createStarted();

final User user = trackAsyncEvent.getUser();
final Track track = trackAsyncEvent.getTrack();

TransactionUtil.handle(() -> {
if (user.fmObject.getLastFmSessionToken() != null) {
    final LastFmService lastFmService = AppContext.getInstance().getLastFmService();
    if(liked)
        lastFmService.loveTrack(user, track);
    else
        lastFmService.unloveTrack(user, track);
}
});
if(liked)
{
    if (log.isInfoEnabled()) {
        log.info(MessageFormat.format( s: "Track liked completed in {0}", stopwatch));
    }
}
else
{
    if (log.isInfoEnabled()) {
        log.info(MessageFormat.format( s: "Track unliked completed in {0}", stopwatch));
    }
}
```

6.) Imperative Abstraction

We merged two classes LastFmUpdateLovedTrackAsyncListener and LastFmUpdateTrackPlayCountAsyncListener into a single class named onLastUpdateTrackPlayCount and similarly LastFmUpdateLovedTrackAsyncEvent and LastFmUpdateTrackPlayCountAsyncEvent into LastFmLovedOrPlayCountTrackAsyncEvent.



```
public void onLastFmUpdateTrackPlayCount(final LastFmUpdateLovedOrPlayCountTrackAsyncEvent lastFmUpdateLovedOrPlayCountTrackAsyncEvent)
    boolean lovedStatus=lastFmUpdateLovedOrPlayCountTrackAsyncEvent.getLovedStatus();
    final User user = lastFmUpdateLovedOrPlayCountTrackAsyncEvent.getUser();
    if(lovedStatus) {
        if (log.isInfoEnabled()) {
            log.info("Last.fm update track play count event: " + lastFmUpdateLovedOrPlayCountTrackAsyncEvent.toString());
        }
        Stopwatch stopwatch = Stopwatch.createStarted();
        TransactionUtil.handle(() -> {
            final LastFmService lastFmService = AppContext.getInstance().getLastFmService();
            lastFmService.importTrackPlayCount(user);
        });
        if (log.isInfoEnabled()) {
            log.info(MessageFormat.format( s: "Last.fm update track play count event completed in {0}", stopwatch));
        }
    }
    else {
        if (log.isInfoEnabled()) {
            log.info("Last.fm update loved track event: " + lastFmUpdateLovedOrPlayCountTrackAsyncEvent.toString());
        }
        Stopwatch stopwatch = Stopwatch.createStarted();
        TransactionUtil.handle(() -> {
            final LastFmService lastFmService = AppContext.getInstance().getLastFmService();
            lastFmService.importLovedTrack(user);
        });
        if (log.isInfoEnabled()) {
            log.info(MessageFormat.format( s: "Last.fm update loved track event completed in {0}", stopwatch));
        }
    }
}
```

```java
public class LastFmUpdateLovedOrPlayCountTrackAsyncEvent {
    3 usages
    private User user;
    2 usages
    private boolean loved;
    4 usages
    public LastFmUpdateLovedOrPlayCountTrackAsyncEvent(User user,boolean loved) {
        this.user = user;
        this.loved=loved;
    }
    public User getUser() { return user; }
    1 usage
    public boolean getLovedStatus() { return loved; }
    @Override
    public String toString() {
        return Objects.toStringHelper( self: this)
                .add( name: "user", user)
                .toString();
    }
}
```

7.) Imperative Abstraction

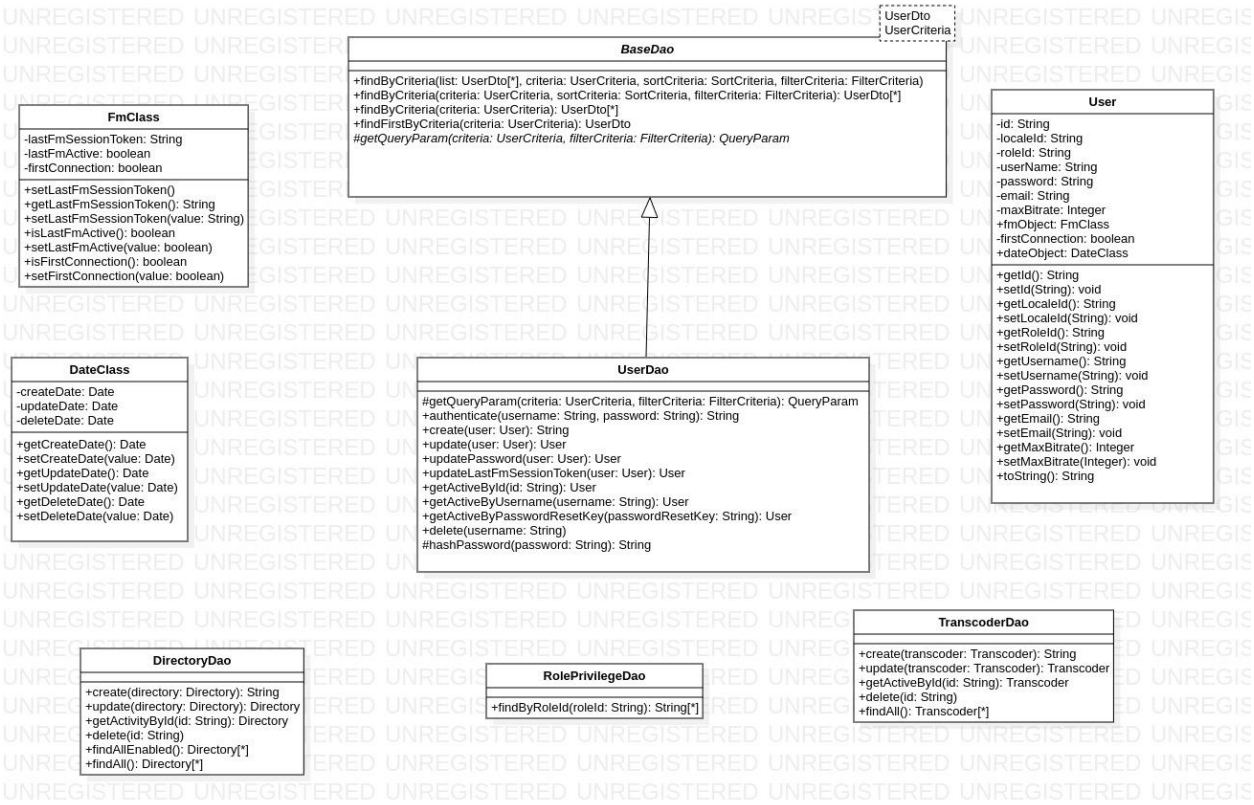
We merged PlayCompletedEvent and PlayStarted event into a single class named PlayEvent and similarly we merged PlayCompletedAsyncListener and PlayStartedAsyncListener into a single class named onPlayStatus.

```java
public class PlayEvent {
    3 usages
    private String userId;
    3 usages
    private Track track;
    2 usages
    private boolean started;
    1 usage
    public PlayEvent(String userId, Track track,boolean started) {
        this.userId = userId;
        this.track = track;
        this.started=false;
    }
    public String getUserId() { return userId; }
    public Track getTrack() { return track; }
    1 usage
    public boolean getStartedStatus(){return started;}
    @Override
    public String toString() {
        return Objects.toStringHelper( self: this)
                .add( name: "userId", userId)
                .add( name: "trackId", track.getId())
                .toString();
    }
}
```

```java
public void onPlayStatus(final PlayEvent playEvent) throws Exception {
    boolean startStatus=playEvent.getStartedStatus();
    final String userId = playEvent.getUserId();
    final Track track = playEvent.getTrack();
    if(startStatus)
    {
        if (log.isInfoEnabled()) {
            log.info("Play started event: " + playEvent.toString());
        }
        TransactionUtil.handle(() -> {
            // Increment the play count
            final User user = new UserDao().getActiveById(userId);
            if (user != null && user.fmObject.getLastFmSessionToken() != null) {
                final LastFmService lastFmService = AppContext.getInstance().getLastFmService();
                lastFmService.scrobbleTrack(user, track);
            }
        });
    }
    else {
        if (log.isInfoEnabled()) {
            log.info("Play completed event: " + playEvent.toString());
        }
        TransactionUtil.handle(() -> {
            // Increment the play count
            UserTrackDao userTrackDao = new UserTrackDao();
            userTrackDao.incrementPlayCount(userId, track.getId());
            final User user = new UserDao().getActiveById(userId);
            if (user != null && user.fmObject.getLastFmSessionToken() != null) {
                final LastFmService lastFmService = AppContext.getInstance().getLastFmService();
                lastFmService.scrobbleTrack(user, track);
            }
        });
    }
```

Refractored UML diagrams

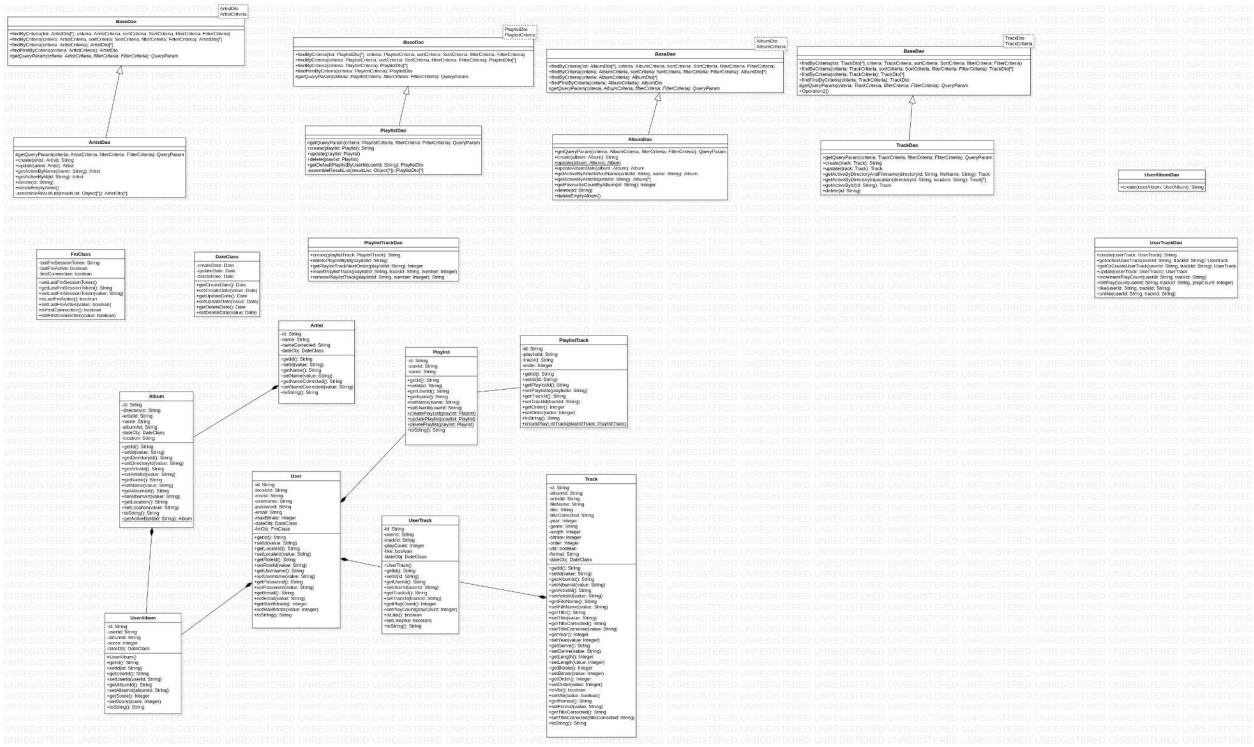
User Management :



If not clear visit:

https://drive.google.com/file/d/1bE-BOCcxzF6ubosIOAMt7iPpgrWiZCO1/view?usp=sharing

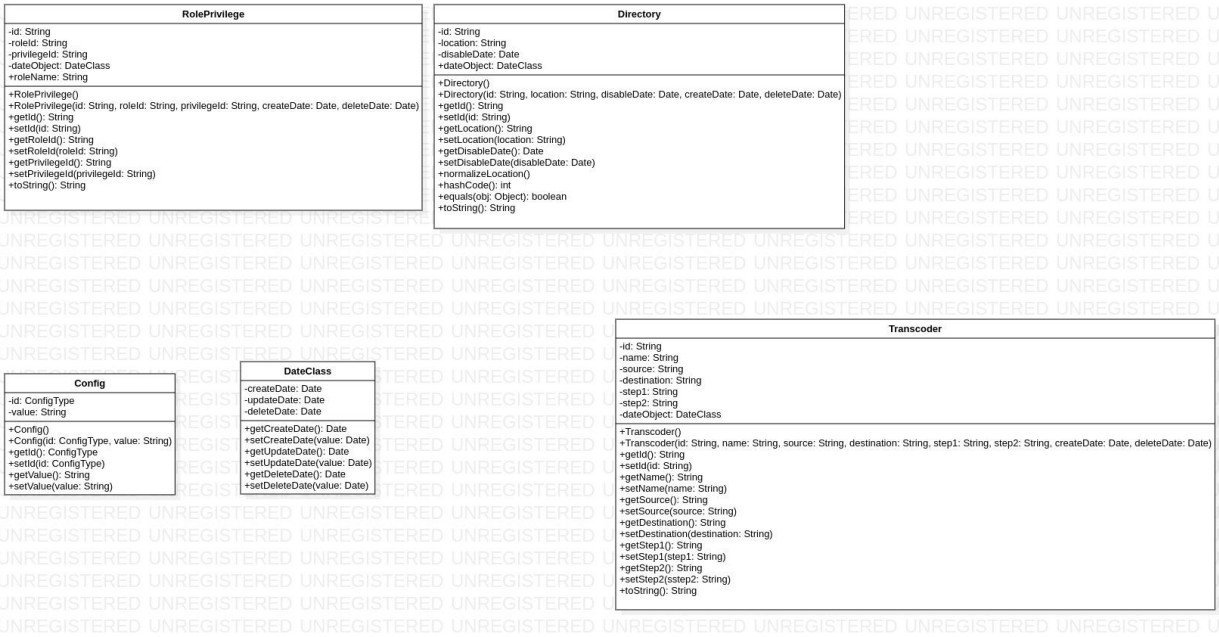
Library Management:



If not clear visit:

https://drive.google.com/file/d/1atPQwAonzV7mGtnu5KC4y9ZRS_elJlBH/view?usp=sharing

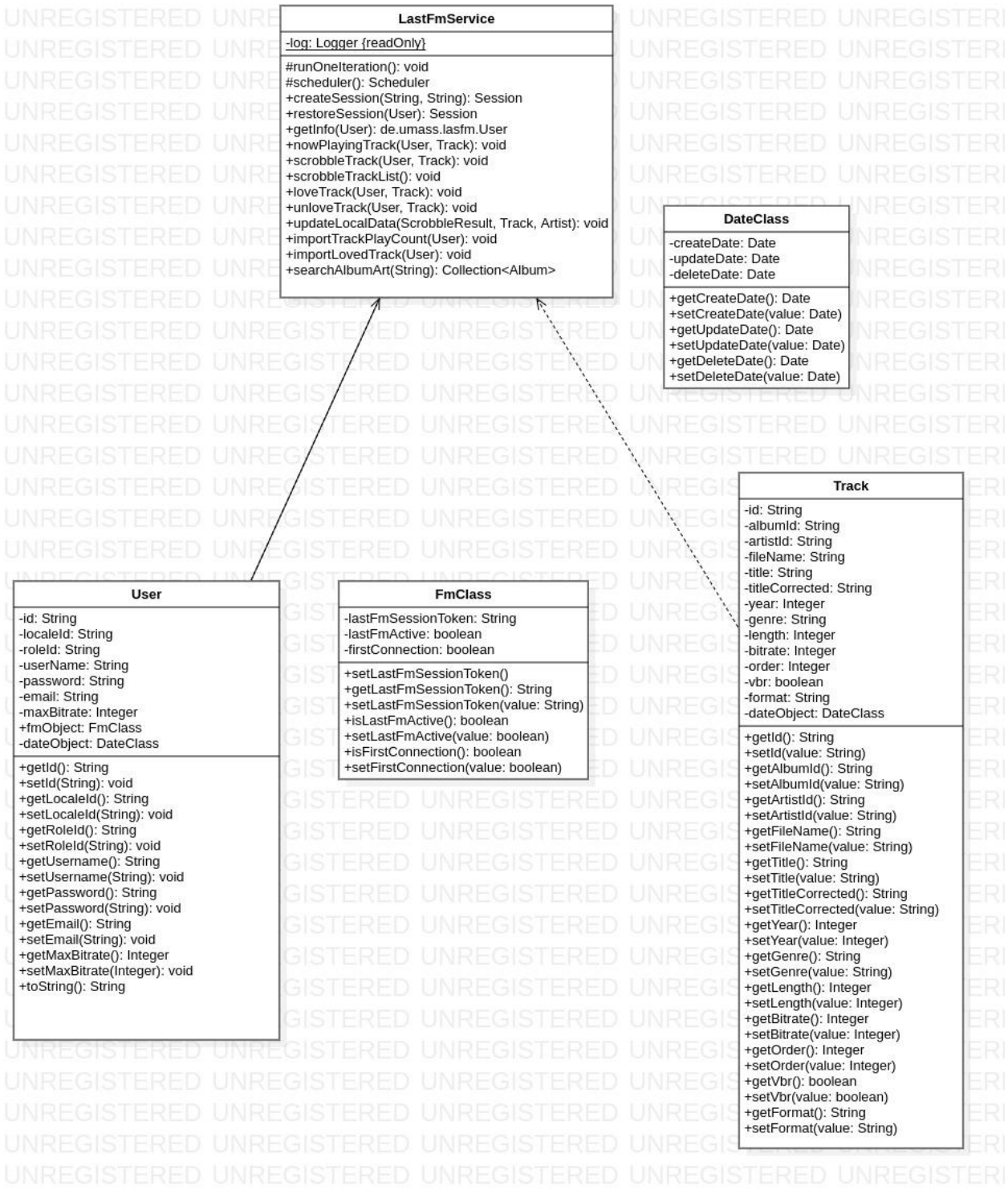
Administration:



If not clear:

an.jpg

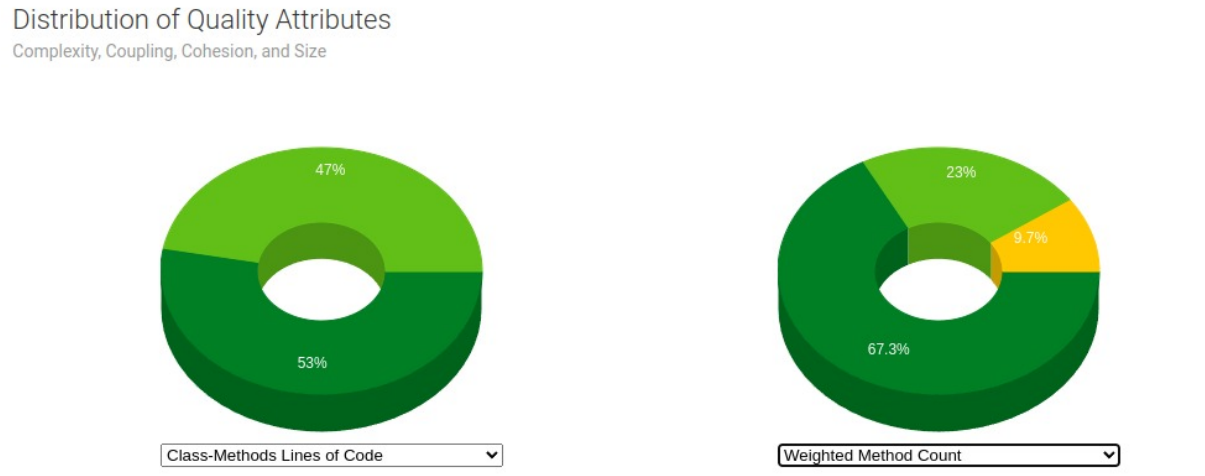
LastFm:



If not clear visit:

https://drive.google.com/file/d/1AbFr1n7kNwr5zuDGoBf1X9DUw-0rWeiF/view?usp=sharing

Task 3-b
Code metrics after refactoring



We found 3 design smells namely Imperative Abstraction, Broken Modularization, and Missing Abstraction. We refactored the code as explained above and eliminated the above-mentioned design smells. Hence, our code metrics improved as described below:

Metric	Before refactoring	After refactoring
Coupling	63	63.2
Complexity	50.7	52.3
Lack of cohesion	72.6	72.8
Size	42.6	43

From the above table we can say that our code quality improved after refactoring.