Projecte de pràctiques – Sessió 7: Preguntes i Comentaris (Part I) –

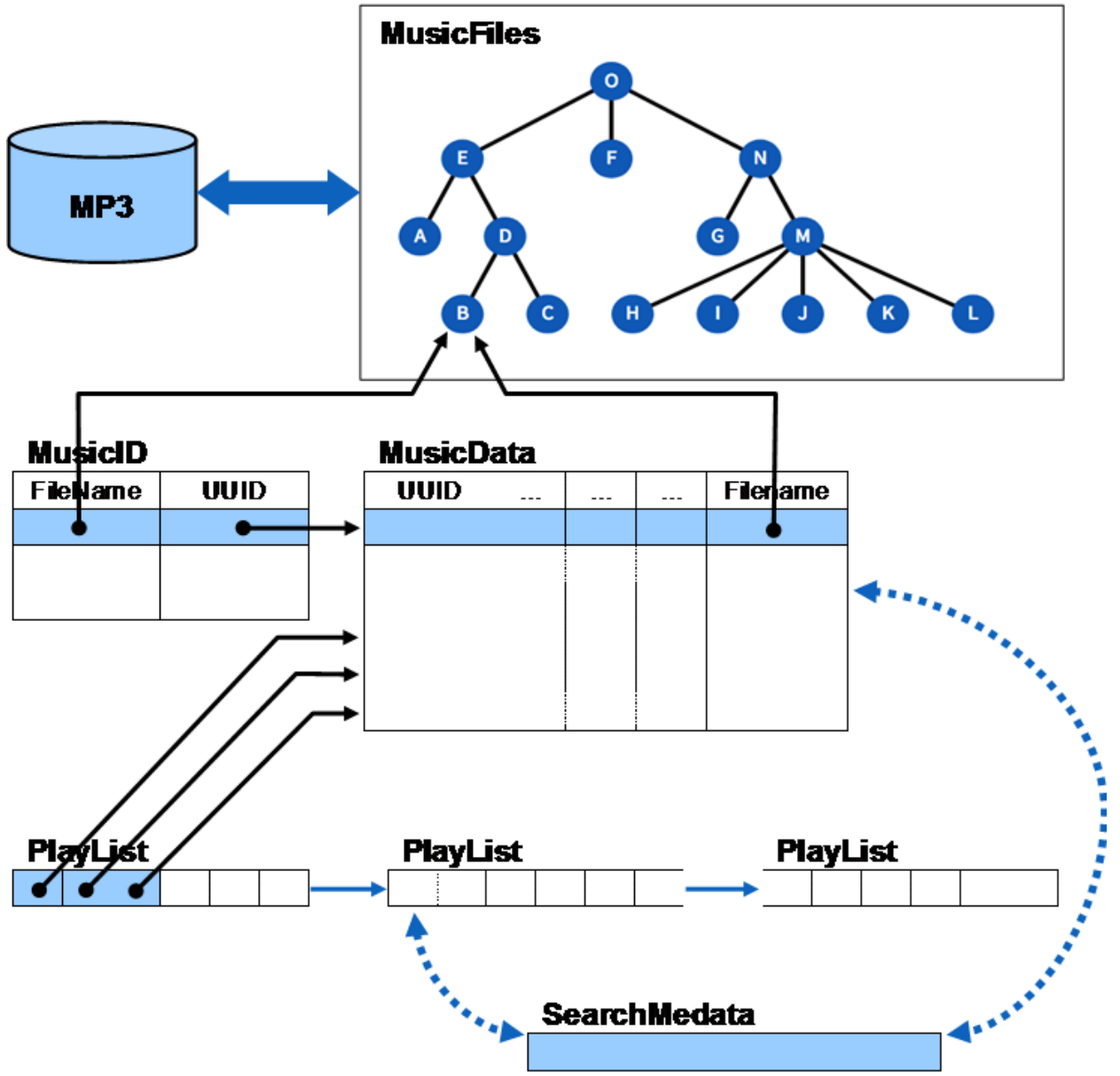
ED

Canvi de **paradigma** respecte d'altres cursos

- **Model de pràctiques fins ara ha estat**:
  - Tot el disseny del projecte vos era proporcionat: classes, mètodes, etc...
  - Només actuàveu com a programadors, doncs no podíeu canviar el disseny.
- **Model de pràctiques des d'ara:**
  - Teniu descrites les funcionalitats que es demanen a la pràctica.
  - En el disseny només estan predefinits alguns mètodes (perquè els necessitem per testejar el codi i també perquè vos orienten en la resolució del problema).
  - Per tant, a partir d'ara actuareu també com a **dissenyadors**, decidint certs aspectes de les classes i de com guardar la informació.
    - *En un primer moment potser el vostre resultat no serà òptim, però durant el curs agafareu les competències per a optimitzar-lo i que el resultat final sigui conforme a les especificacions donades.*

Diagrama general

- Diferents classes encarregades de diverses funcionalitats.
- Cada objecte gestiona les seves pròpies dades.
- Cal implementar les interfícies mínimes comunes descrites.
- Segons les necessitats caldrà afegir diferents mètodes que serviran per a compartir informació.



Objectius del projecte

- **Resum dels objectius de la Part I de la pràctica**:
  - Tenir-ne un entorn funcional de desenvolupament en el domini del problema.
  - Implementar la funcionalitat interna bàsica dels diferents components.
    - Existirà (en principi) un baix acoblament entre les parts.
  - Centrar-se en els objectius funcionals sense preocupar-se del rendiment.
    - La implementació de les estructures de dades no cal que sigui òptima.
    - Es poden reutilitzar estructures de Python lliurement.
  - Garantir un funcionament determinista fent servir un conjunt de proves.
    - Cal assegurar-se de completar els tests de les diferents classes.
  - Documentar correctament els vostres programes.
    - Afegir comentaris dins el codi i complementar-lo amb un document adjunt.

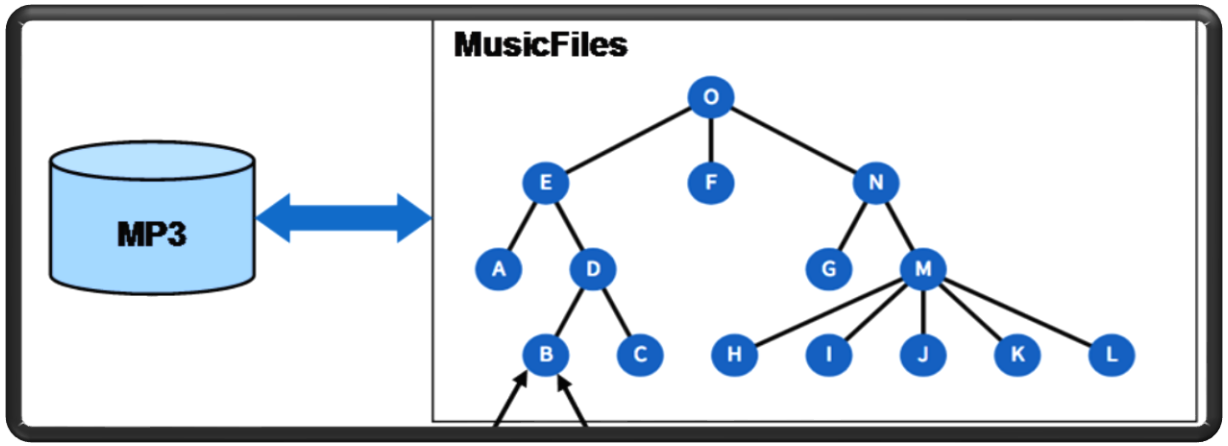
Metodologia d'aquesta sessió

- El principal problema darrera la majoria de les qüestions plantejades es relaciona amb una comprensió incorrecta o incompleta de les funcionalitats requerides dins cadascuna de les classes demanades.
- Per tant, analitzarem cada funcionalitat/classe en detall, descrivint els detalls més rellevants.

Podeu fer preguntes en qualsevol moment!

Func1 – Class MusicFiles

- **OBJECTIU**: Guardar la col·lecció dels arxius MP3.
- **RESPONSABILITAT**: Revisar el contingut dels disc per a obtenir les adreces dels MP3 que actualment existeixen dins la col·lecció de música.

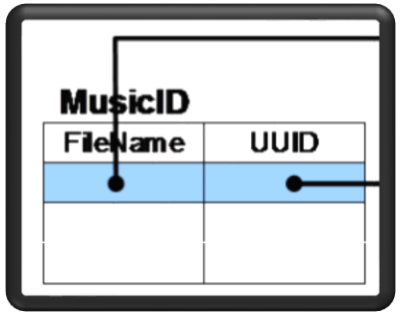


- Físicament els arxius MP3 estaran exclusivament en el disc, per tant la classe només guarda el path dels arxius.
- El ROOT_DIR identifica des d'on s'han de començar a cercar els MP3, essent necessari revisar tots els subdirectoris amb els seus propis subdirectoris.
- Utilitzant el paquet *os.path* podeu saber la llista d'arxius.
- La interfície descrita implica que es cerquen els arxius en un moment determinat amb *reload_fs()*; i que es pot tornar a revisar el disc en qualsevol moment tornant a cridar a aquesta mateixa funció.
- Per tant, és després de cridar *reload_fs()* que si ja hi havia arxius a la col·lecció llavors és quan té sentit cridar a les altres dues funcions per a conèixer els canvis.

- `MusicFiles.reload_fs(path: str)`
- `MusicFiles.files_added() -> list`
- `MusicFiles.files_removed() -> list`

Func2 – Class MusicID

- **OBJECTIU**: Gestionar el parell *file-path <-> UUID*.
- **RESPONSABILITAT**: Generar identificadors únics (UUID) per a cada arxiu MP3 individual.

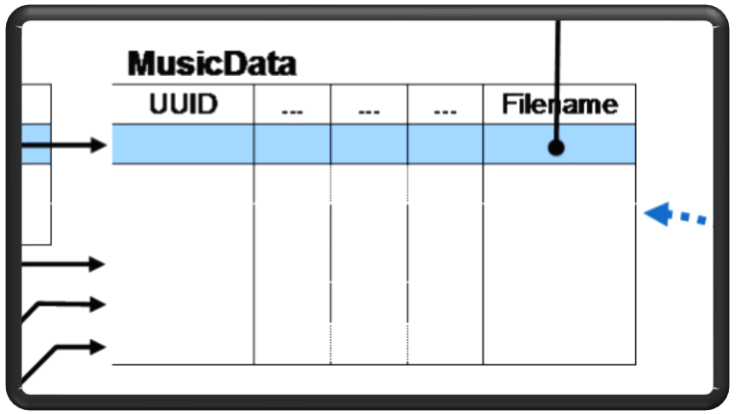


- El UUID és com una impremta digital del MP3, en aquest cas basada en el file-path de l'arxiu (no pas en el seu contingut).
- El UUID té una mida fixa, raó per la qual és més adient de fer-lo servir que no pas el file-path.
- Ja que la funció que genera els UUID no pot garantir al 100% les col·lisions cal comprovar i gestionar els valors creats.
- A l'exemple del projecte podeu veure com generar el UUID fent servir la funció *uuid.uuid5(uuid.NAMESPACE_URL, path)*.
- Degut a que el valor generat depèn del string d'entrada, és adient calcular-lo a partir d'un file-path canònic, és a dir, una representació independent de la plataforma. Cal fer servir la funció dins "cfg.py" anomenada *get_canonical_pathfile()*, tenint en compte que el resultat no inclourà el ROOT_DIR.

- `MusicID.generate_uuid(file: str) -> str`
- `MusicID.get_uuid(file: str) -> str`
- `MusicID.remove_uuid(uuid: str)`

Func3 – Class MusicData

- **OBJECTIU**: Guardar les metadades dels arxius MP3 llegint-les un sol cop per a cada arxiu.
- **RESPONSABILITAT**: Per a cada arxiu afegit, ha de guardar el seu UUID (clau/índex), les metadades que contingui l'arxiu MP3 i el file-path.



- Les metadades estan dins els arxius MP3 i es recuperen fent servir una llibreria (eye3d). Seguiu l'exemple per a fer-ho correctament.
- Els mètodes *add_song()* i *remove_song()* gestionen la creació/esborrament de les entrades dins l'estructura de dades que vosaltres fareu servir.
- La funció *load_metadata()* es pot cridar quan el UUID ja està inserit, i serveix per a carregar les dades des del MP3.
- No tots els MP3 contenen metadades, o les tenen completes. Les dades poden ser incomplertes. Per tant, no cal preocupar-se dels *warnings* que genera la llibreria respecte als continguts quan processa un arxiu.

- `MusicData.add_song(uuid: str, file: str)`
- `MusicData.remove_song(uuid: str)`
- `MusicData.load_metadata(uuid: str)`
- `MusicData.get_title(uuid: str) -> str`
- `MusicData.get_artist(uuid: str) -> str`
- `MusicData.get_album(uuid: str) -> str`
- `MusicData.get_genre(uuid: str) -> str`

Func4 – Class MusicPlayer

- **OBJECTIU**: Reproduir les cançons dels arxius MP3.
- **RESPONSABILITAT**: Utilitzar d'altres classes per a completar la seva funcionalitat de reproduir cançons.

- Aquesta classe no guarda informació, només realitza les operacions demanades.
- És un nexe que connecta altres classes dins el programa.
- La funció reproduir pot ser pels altaveus+pantalla, o només per una de les dues sortides. Això es selecciona amb el paràmetre *mode* del mètode *play_song()*.
- Els mètodes *print_song()* i *play_file()* són bàsicament mètodes interns de la classe.
- Si vos resulta necessari podeu definir nous mètodes dins les altres classes per a que aquesta pugui completar la seva funcionalitat; però sigueu prudents de mantenir la encapsulació de les dades.
- El mode 0 és el que realment s'utilitzarà dins l'avaluació.

- `MusicPlayer.print_song(uuid: str)`
- `MusicPlayer.play_file(file: str)`
- `MusicPlayer.play_song(uuid: str, mode: int)`

```
mode : 0 printem cançó per pantalla
mode : 1 printem cançó i fem play
mode : 2 play de la cançó però no printem
```

Func5 – Class PlayList

- **OBJECTIU**: Guardar una llista de cançons.
- **RESPONSABILITAT**: Crear una llista de cançons i guardar els UUID d'aquestes cançons. També es poden reproduir els arxius que hi ha dins la llista.

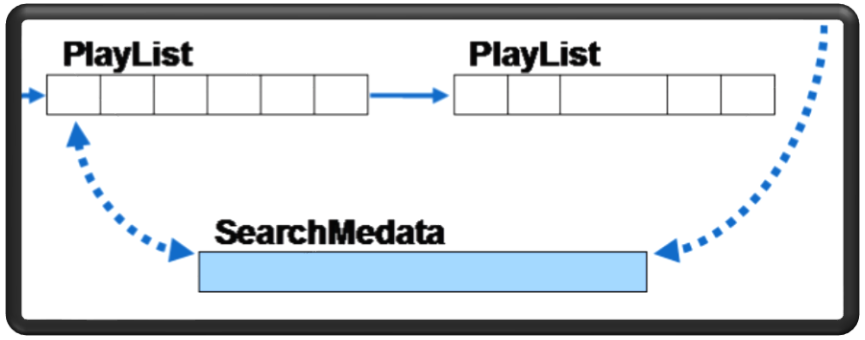
- Cada instància de la classe és una llista.
- Com que existeixen múltiples llistes, cal enllaçar-les.
- Les llistes es llegeixen des d'arxius M3U.
- Els arxius M3U son línies de text, que contenen el file-path *relatiu* dels arxius MP3, els quals caldrà comprovar que hi són a la col·lecció i determinar el seu UUID.
- El mètode *play()* utilitzarà la classe MusicPlayer per a reproduir les cançons existents a la PlayList.

- `PlayList.load_file(file: str)`
- `PlayList.play()`

Per a informació respecte al format M3U consulteu: https://en.wikipedia.org/wiki/M3U#Examples

Func6 – Class SearchMetadata

- **OBJECTIU**: Realitzar cerques que generen llistes de cançons.
- **RESPONSABILITAT**: Consulta les metadades dins la col·lecció musical i proporcionar funcions per a manipular llistes de cançons.



- Una llista de cançons és semànticament una PlayList, però no ho és dins el programa. Els paràmetres que manipula aquesta classe són Python lists de UUIDs.
- Aquesta classe ha d'utilitzar mètodes de la classe MusicData per a fer les consultes a les cerques.
- Així doncs per a utilitzar la classe MusicData amb la interfície definida cal poder iterar per tota la col·lecció d'elements. Així doncs cal dissenyar i definir una estratègia per a fer-ho.

- `SearchMetadata.title(sub: str) -> list (de UUIDs)`
- `SearchMetadata.artist(sub: str) -> list (de UUIDs)`
- `SearchMetadata.album(sub: str) -> list (de UUIDs)`
- `SearchMetadata.genre(sub: str) -> list (de UUIDs)`
- `SearchMetadata.and_operator(list1: list, list2: list) -> list (de UUIDs)`
- `SearchMetadata.or_operator(list1: list, list2: list) -> list (de UUIDs)`

Func7 – Class PlayList

- **OBJECTIU+**: Guardar una llista de cançons com a PlayList.
- **RESPONSABILITAT+**: Inserir i treure elements d'una PlayList.

- A més de poder llegir una llista de cançons des d'un arxiu M3U, té sentit poder convertir una cerca (*list[UUID]*) en una instància de PlayList.
- Per a poder fer això s'afegeix aquesta interfície a PlayList.
- Conceptualment aquesta funcionalitat afegida serveix simplement per a "guardar" els resultats d'una cerca en una PlayList.

- `PlayList.add_song_at_end(uuid: str)`
- `PlayList.remove_first_song()`
- `PlayList.remove_last_song()`

Recordeu: https://en.wikipedia.org/wiki/M3U#Examples

Comentaris finals

- Recordeu afegir "*import cfg*" a l'inici de cada un dels vostres arxius de codi font.
- Feu servir el mateix nom de la classe pels arxius Python, doncs així cal fer-ho al Caronte per a les proves.
- Consulteu la documentació de Python per a trobar mètodes que facin determinades accions que pugueu necessitar dins el vostre codi (no cal refer la roda).
- Sou lliures (dins la part I del projecte) de fer servir qualsevol estructura ja implementada dins Python per a guardar les dades que necessiteu.
- No cal preocupar-se (ara) del rendiment de les estructures de dades que feu servir, però sí de la seva funcionalitat (validesa).
- Recordeu fer les proves que verifiquin el vostre codi.
- El codi de setup 'P0.ZIP' cal tenir-ho ja instal·lat.