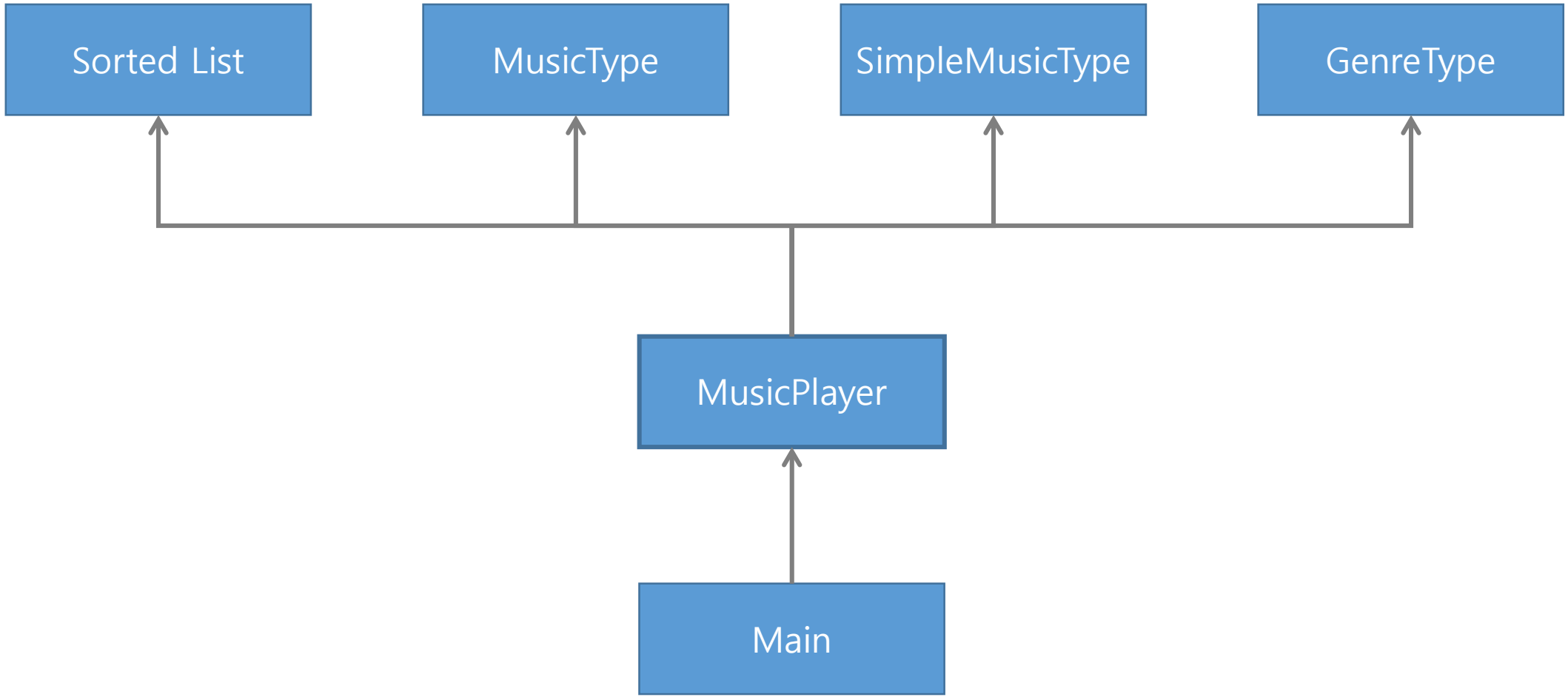
프로그래밍 과제 1을 위한 간단한 구조 설명

2018.10.02

개요

- 본 자료에서는 프로그래밍 과제 1을 위한 간단한 구조를 설명한다.
- 설명을 위해 가장 간단한 스켈래톤 코드를 제공한다.
- 제공 된 코드를 참고하여 프로그래밍 과제 1을 스스로 완성해야한다.

Class Diagram



Class info : MusicPlayer

- Application과 같은 역할을 하는 프로그램의 가장 겉부분
- 모든 음악정보를 관리하는 mMusicList와 장르 별로 관리하는 mGenreList가 있다.

```
protected:
    SortedList<MusicType> mMusicList;
    SortedList<GenreType> mGenreList;

    int m_Command;
};
```

Class info : MusicPlayer

```
int MusicPlayer::AddMusic()
{
    MusicType indata1;
    SimpleMusicType indata2;

    indata1.setRecordFromKB(); // mMusicList에 넣을 모든 음악 정보
    indata2.setInfo(indata1.getId(), indata1.getTitle());   // mGenr

    // MusicList에 추가
    mMusicList.Add(indata1);

    // GenreList에 추가
    GenreType temp; // 입력된 음악이 어느 장르에 속하는지 찾기 위한
    GenreType *pData; // mGenreList의 해당 장르를 포인터로 가르키기

    mGenreList.ResetList();
    int length = mGenreList.GetLength();
    int curIndex = mGenreList.GetNextItem(temp);
    while (curIndex < length && curIndex != -1)
    {
        if (temp.getGenreName().compare(indata1.getGenre()) == 0)
        {
            pData = mGenreList.GetPoint(temp); // 입력된 음악과 일치
            pData->AddMusicInGenre(indata2); // 리스트 안 리스트에 음
            break;
        }
        curIndex = mGenreList.GetNextItem(temp);
    }


    return 1;
}
```

int AddMuic()

- 새로운 음악을 추가하는 함수
- 음악을 추가함과 동시에 mMusicList와 mGenreList 모두에게 새로운 음악 정보를 추가한다.

Class info : MusicPlayer

```
int MusicPlayer::DisplayMusicInGenre()
{
    GenreType data;

    cout << "\n\tCurrent list" << endl;

    // list의 모든 데이터를 화면에 출력
    mGenreList.ResetList();
    int length = mGenreList.GetLength();
    int curIndex = mGenreList.GetNextItem(data);
    while (curIndex < length && curIndex != -1)
    {
        cout << curIndex + 1 << endl;
        data.DisplayAll();
        curIndex = mGenreList.GetNextItem(data);
    }

    // 특정 장르의 음악 정보만 보고 싶을 때
    int id;
    GenreType *pData;

    cout << "Select Genre number : ";
    cin >> id;
    data.SetInfo(id, "");
    pData = mGenreList.GetPoint(data);
    pData->SearchMusicDetailInGenre(&mMusicList); // 해당 장르 음악 출력

    return 1;
}
```

int DisplayMusicInGenre()

- mGenreList에서 장르를 선택 한 후 해당 장르에 속하는 모든 음악 정보를 출력하는 함수

Class info : SortedList

- 정렬된 Array 리스트
- template이기 때문에 어떤 타입이 오더라도 Generic하게 작성되어 있어야 한다.

  그렇기 때문에 기존의 CompareByID()나 GetId() 등의 함수는 사용 불가

```
if(CurItem CompareByID(inData)==GREATER | m_CurPointer==m_Length)  /
{
    for(int i=m_Length;i>m_CurPointer;i--)  //맨뒤에서 부터 하나씩 땡
        m_Array[i]=m_Array[i-1];    //배열 밀기
    m_Array[m_CurPointer]=inData;   //배열 밀은 후 현재 포인터에 아이
    m_Length++; //개수 증가
    break;
}
GetNextItem(CurItem);   //다음아이템으로
```

```
if (m_Array[i].GetId() == data.GetId()) //da
{
    for (int j = i; j<m_Length - 1; j++)
    {
        m_Array[j] = m_Array[j + 1];
    }
    m_Length--; //지울 항목의 자리에 다음 항
    return 1;   //항목을 삭제하는데 성공했으
}
```

따라서 비교 연산자 Overloading을 사용해야한다.

Class info : MusicType

- 모든 음악 정보를 가진 ItemType
- 음악ID(Primarykey), 음악명, 가수, 앨범, 장르를 가지고 있다.

```
protected:
    int mMusicId;   ///< Primary key
    std::string mTitle;
    std::string mSinger;
    std::string mAlbum;
    std::string mGenre;
```

- 비교연산자 오버로딩과 그 외 필요한 함수는 본인이 작성

```
// complete operation overloadings...
bool operator== (const MusicType &obj) { return 0; }
bool operator> (const MusicType &obj) { return 0; }
bool operator< (const MusicType &obj) { return 0; }
bool operator>= (const MusicType &obj) { return 0; }
bool operator<= (const MusicType &obj) { return 0; }
```

Class info : SimpleMusicType

- 일부 음악 정보를 가진 ItemType
- 음악ID(Primarykey), 음악명을 가지고 있다.

```
protected:
    int mMusicId;    ///< Primary key
    std::string mTitle;
```

- 불필요한 데이터낭비를 막기 위해 일부 음악정보만 가질 수 있는 Type이 필요하다.
- 비교연산자 오버로딩과 그 외 필요한 함수는 본인이 작성

```
// complete operation overloadings...
bool operator== (const SimpleMusicType &obj) { return 0; }
bool operator> (const SimpleMusicType &obj) { return 0; }
bool operator< (const SimpleMusicType &obj) { return 0; }
bool operator>= (const SimpleMusicType &obj) { return 0; }
bool operator<= (const SimpleMusicType &obj) { return 0; }
```

Class info : GenreType

- 장르를 가지고 있는 ItemType
- 장르ID(Primarykey), 장르명, 해당 장르에 속한 음악 리스트을 가지고 있다.

```
protected:
    int mGenreId;    ///< Primary key
    std::string mGenre;

    SortedList<SimpleMusicType> mGenreMusicList;
```

- 비교연산자 오버로딩과 그 외 필요한 함수는 본인이 작성

```
// complete operation overloadings...
bool operator== (const GenreType &obj) { return 0; }
bool operator> (const GenreType &obj) { return 0; }
bool operator< (const GenreType &obj) { return 0; }
bool operator>= (const GenreType &obj) { return 0; }
bool operator<= (const GenreType &obj) { return 0; }
```

Class info : GenreType

- Type 내부에 리스트 안 리스트에 접근할 수 있는 함수를 지니고 있다.
- void AddMusicInGenre(SimpleMusicType indata)

  : mGenreMusicList에 일부 음악 정보를 추가하는 함수

```
void GenreType::AddMusicInGenre(SimpleMusicType indata)
{
    mGenreMusicList.Add(indata);
}
```

- 위 Add 함수 이외 Delete, Replace, Search 함수들 도한 동일하게 구현할 수 있다.
- void SearchMusicDetailInGenre(SortedList<MusicType> *list)

  : 해당 장르에 속하는 음악 정보를 볼 수 있는 함수

  파라미터의 의미는 스스로 잘 생각보도록 하자

```
void GenreType::DisplayMusicDetailInGenre(SortedList<MusicType> *list)
{
    SimpleMusicType data;

    cout << "₩n₩tCurrent list" << endl;

    // list의 모든 데이터를 화면에 출력
    mGenreMusicList.ResetList();
    int length = mGenreMusicList.GetLength();
    int curIndex = mGenreMusicList.GetNextItem(data);
    while (curIndex < length && curIndex != -1)
    {
        cout << curIndex + 1 << endl;
        data.DisplayAll();
        curIndex = mGenreMusicList.GetNextItem(data);
    }

    //?? SimpleMusicType 인 data 는 music id 와 title 만을 갖는다.
    // 해당 장르에 포함되는 모든 음악 정보를 출력하기 위해서 이 정보를 가지고
    // 이 함수의 파라미터인 list를 검색하여 music information 을 출력해 주어야 함.
}
```