Project 2

Programação - MIEIC (2012/2013)

This document contains the first part of the project.

Each class is separated by a pagebreak, and has a brief description at the top.

Each class is commented where we felt comments were required.

Eduardo Almeida, João Almeida and João Ramos - MIEIC (2012/2013)

```
//
//  User.h
//  Created by Eduardo Almeida, Joao Almeida and Joao Ramos
//
//  This class takes care of the User objects and their
information.
//

#include <iostream>
#include <string>
#include <vector>

class Playlist;

typedef enum {
    kSexMale,
    kSexFemale
} kSex;

class User {

private:

    unsigned int userId;
    unsigned int age;

    kSex gender;

    std::string name;

    bool isAdministrator;

    Playlist userPlaylist;

public:

    User();
    ~User();

    bool setUserId(unsigned int theId);
    const unsigned int userId();

    bool setGender(kSex theGender);
    const kSex gender();

    bool setName(std::string theName);
    const std::string name();
    bool setAsAdmin(bool isAdmin);
    const bool isAdmin();

    bool setPlaylist(Playlist thePlaylist);
    const Playlist playlist();

};
```

```
//
//  UserManager.h
//  Created by Eduardo Almeida, Joao Almeida and Joao Ramos
//
//  This class manages the different users and their permissions.
//

#include <iostream>
#include <string>
#include <vector>

class User;

class UserManager {

private:

    UserManager(); // There can only be one user manager, so a 
singleton here is completely appropriate.

    UserManager(UserManager const&); // Copy constructor is 
private.
    UserManager& operator=(UserManager const&); // Assignment 
operator is private.

    static UserManager *um_pInstance;

    std::vector<User *> userVector;

public:

    static UserManager* Instance();

    bool addUser(User *aUser);

    bool removeUser(User *aUser);
    bool removeUser(int userId);

    const User* getUser(int userId);
    const User* getUser(std::string userName);

    const unsigned int userCount();
    const unsigned int adminCount();

};
```

```
//
//  Music.h
//  Created by Eduardo Almeida, Joao Almeida and Joao Ramos
//
//  This class manages a single song object and its data.
//

#include <iostream>
#include <string>

class Music {

private:

    unsigned int musicId;

    unsigned int year;

    unsigned int likes;
    unsigned int dislikes;

    unsigned int playCount;

    bool available;

public:

    // Doing this with constructors and deconstructors is much 
better as we can
    // automatically fill the song id by accessing the radio music 
database.

    Music();
    ~Music();

    //
    // User variables.
    //

    bool setMusicId(unsigned int theId);
    const unsigned int musicId();

    bool setYear(unsigned int theYear);
    const unsigned int year();

    const unsigned int likes();
    const unsigned int dislikes();
    bool addLike();
    bool addDislike();

    const unsigned int playCount();
    bool addPlay();

    const bool available();
    bool setAvailable(bool availability);

};
```

```
//
//  Playlist.h
//  Created by Eduardo Almeida, Joao Almeida and Joao Ramos
//
//  This class manages a single playlist object and its songs/song
count.
//

#include <iostream>
#include <vector>
#include <map>

class Music;

class Playlist {
private:
    std::vector <Music> thePlaylist;

public:
    //
    // Add and remove songs to the playlist, nothing major here.
    //

    void addSong(Music *theSong, int playCount);
    void removeSong(Music *theSong);

    //
    // As we are using the Music objects by reference, and using
    // the same object on the global/user/... playlists, we can
    // generate the top ten songs list dynamically.
    //

    // Returns Music * as the key and the song count as the
object.

    // As maps are ordered, we can just loop through this.

    const std::map <Music *, int> topTenSongs();

    // Searches through the playlist and returns a vector with
matches.

    const std::vector<Music *> search(std::string title, int year,
std::string artist, std::string music_genre);
};
```

```
//
//  FilesIO.h
//  Created by Eduardo Almeida, Joao Almeida and Joao Ramos
//
//  This class manages everything that has to do with files.
//

#include <iostream>
#include <vector>
#include <string>

class Music;
class Playlist;

class FilesIO {

private:

    FilesIO(); // We hope we aren't using too many singletons!

    FilesIO(FilesIO const&); // Copy constructor is private.
    FilesIO& operator=(FilesIO const&); // Assignment operator is
private.

    static FilesIO *fio_pInstance;

public:

    static FilesIO* Instance();

    //
    // The global files
    //   - radioStationMusics.csv
    //   - topTen.csv
    //   - users.csv
    //
    // Usually only need to be loaded / written to
    // a few times per session.
    //
    // So we can merge them, for simplicity.
    //

    bool loadGlobals();
    bool storeGlobals();

    //
    // Loads the playlistUserXXX.csv or searches
    // for an user with that username.
    //

    Playlist playlistForUser(int userid);
    Playlist playlistForUser(std::string userName);

    //
    // Saves the user playlist.
    //

    bool storePlaylistForUser(int userid);

};
```