

# Mandatory Assignment 1

## The CORBA Taste Profile Service

By:

Paul Joakim Oulie Andreassen

Thomas Kristiansen

Uy Viet Tran

# How to run the application

1. Start orbd with given port:

Linux: `orbd -ORBInitialPort <Port>`

Windows: `start orbd -ORBInitialPort <Port>`

2. Start server:

`java -jar <Full path to jarfile> -ORBInitialPort <Port> <Full path to input file (the "database file")> <Integer value to decide if cache is to be used on server or not (0 = cache off. 1 = cache on)>`

For example:

`java -jar C:\Users\server.jar -ORBInitialPort 6000 C:\Users\train_triplets.txt 0`

3. Wait until server outputs "Init finished"

4. Start client:

`java -jar <Full path to jarfile> -ORBInitialPort <Port> <Full path to input file> <Full path to output file (including name of file with extension)> <Integer value to decide if cache is to be used on client or not (0 = cache off. 1 = cache on)>`

For example:

`java -jar C:\Users\client.jar -ORBInitialPort 6000 C:\Users\input.txt C:\Users\output.txt 0`

# Design and Implementation

## Client:

The client consist of just one pretty straight forward class. This class connects the client to the server trough the CORBA ORB, and uses this connection to call the different methods available.

What methods to be called are decided by an input file. The class takes three arguments (in addition to the CORBA arguments). The first argument is a path to an input file. It is dependent on this file to know what methods to call. The input file has to consist of one or more lines, where each line holds information about the method to call and the parameters to pass to the method. The second argument is a path to an output file. This is the file where the values returned from the methods are outputted to. The last argument is an integer, which decides if client side cache is to be used or not (0 = client cache off, 1 = client cache on). More info about the cache is written under the “Cache” heading below.

```
C:\Users\Bruker\OneDrive\UiO\IN5020 Distribuerte systemer\Assignments\IN5020-CORBA>java -jar client.jar -ORBInitialPort 1050 "C:\Users\Bruker\OneDrive\UiO\IN5020 Distribuerte systemer\Assignments\IN5020-CORBA\input.txt" "C:\Users\Bruker\OneDrive\UiO\IN5020 Distribuerte systemer\Assignments\IN5020-CORBA\output.txt" 1
Song SOPSOHT12A67AE0235 played 13 times by user 55874081c91a71d9f7a13cd9e9f1538e23874370. (152 ms)
Song SOPAYPV12AB0170B0C played 6 times by user 55874081c91a71d9f7a13cd9e9f1538e23874370. (1 ms)
Song SONQCKC12A6D4F6A37 played 22 times by user 55874081c91a71d9f7a13cd9e9f1538e23874370. (1 ms)
Song SORVEI12A6701E7F8 played 1 times. (82 ms)
Song SOEGIVH12A6D4FC0E3 played 389880 times. (82 ms)
Song SOJHZAX12AB017E5D2 played 13 times by user a17766790b36cb899f152a083389c3111b7ced61. (6739 ms)
Song SOIRJL012AB0188F2C played 303 times. (82 ms)
Song SOAIWY112A81C206F1 played 648239 times. (82 ms)
Song SOFSITK12AB0182CC7 played 25 times. (86 ms)
Song SOVLWV12AB01849A8 played 1 times by user 0a4f6ff962cc4d343f77502d743b51a09541ead1. (126 ms)
Song SOUDSFN12A8C144B74 played 1069 times. (82 ms)
Song SOIPTH12A8C141954 played 11205 times. (83 ms)
Song SONKFWL12A6D4F93FE played 2 times by user b64cdd1a0bd907e5e00b39e345194768e330d652. (96 ms)
Song SOWTZHU12AB017EADB played 432 times. (84 ms)
Song SOALOU12A6D4F87D2 played 92 times. (83 ms)
Song SOSVTJ12AB01872DA played 1070 times. (82 ms)
Song SOYGGJ12A67AE0ACD played 14 times by user 0025bfe6248070545d23721083acd3f60451da4f. (96 ms)
Song SOUELOC12AB0182DD1 played 2686 times. (84 ms)
Song SOEPZQS12A8C1436C7 played 63951 times. (83 ms)
Song SOZCGFC12A67AE0DFA played 13 times by user 0025bfe6248070545d23721083acd3f60451da4f. (1 ms)
Song SOQTACU12A8C135CB7 played 16 times by user dd0a2cc711c7ae2deb57c0089ded533db473d5b1. (98 ms)
Song SOSUZFA12A8C13C0A4 played 11341 times. (82 ms)
Song SOOWNL012A6D4F7A3C played 4573 times. (84 ms)
Song SOVPSKL12A670206B9 played 31277 times. (83 ms)
Song SOUNZHU12A8AE47481 played 158636 times. (82 ms)
Song SKLRPJ12A8C13C3FE played 128837 times. (82 ms)
Song SOPTXLA12A8C1452D7 played 10 times by user dd0a2cc711c7ae2deb57c0089ded533db473d5b1. (1 ms)
Song SOVWADY12AB0189C63 played 78443 times. (86 ms)
Song SOSZJFV12AB01878CB played 82819 times. (83 ms)
Song SONHVVE12AB018D038 played 72259 times. (84 ms)
Song SORMLTW12A670208FA played 56992 times. (82 ms)
Song SOAKIMP12A8C130995 played 1 times by user b80344d063b5ccb3212f76538f3d9e43d87dca9e. (85 ms)
Song SOGPGNG12A8C143969 played 59295 times. (87 ms)
Song SOYGHUM12AB018139C played 42348 times. (83 ms)
Song SOXPDQ12A58A76829 played 35629 times. (85 ms)
Song SOEWMNK12A67AD8797 played 4 times by user d66f2f66f2bdc9aa3d0362a35fc91ccc844101f7. (92 ms)
Song SOAUNAX12AB01876D0 played 3 times by user efd4f99afd9a2da1ec7c770b6b50f58945a7d581. (11202 ms)
Song SOQQGB812A6D4FCE33 played 23841 times. (83 ms)
Song SOMTLPL12A6702085A played 18168 times. (82 ms)
Song SOBDVAK12AC90759A2 played 3 times by user efd4f99afd9a2da1ec7c770b6b50f58945a7d581. (1 ms)
Song SOBHAGV12A8C142CA6 played 1 times by user efd4f99afd9a2da1ec7c770b6b50f58945a7d581. (1 ms)
Song SOEBYTE12A8C14389C played 17897 times. (83 ms)
Song SOHTRWF12A8C13E4EB played 14029 times. (83 ms)
Song SOSXDX012AF72A1F32 played 3358 times. (83 ms)
Song SOZJUSJ12A8C13C249 played 5120 times. (83 ms)
Song SODLYOY12AB017C6D0 played 2 times by user e3d45fa4071c85b044191cabf0284e74c744a037. (112 ms)
Song SODMSFL12AB0181E6F played 2 times by user e3d45fa4071c85b044191cabf0284e74c744a037. (1 ms)
```

*Screenshot of client running*

## Server and servant:

The server basically consist of two classes. One class that initiates the server through the CORBA ORB, and another class that acts like the servant of the server which holds all methods that are reachable from clients. In addition to the server and servant classes, the server also uses a file holding a huge database of user and song associations. This is the file the different methods of the server uses to get the values it is going to return. The server class takes two arguments (in addition to the CORBA arguments). The first argument is a path to the database file. The second argument is an integer, which decides if server side cache is to be used or not (0 = client cache off, 1 = client cache on). More info about the cache is written under the "Cache" heading below.

```
C:\Users\Bruker\OneDrive\UiO\IN5020 Distribuerte systemer\Assignments\IN5020-CORBA>java -jar server.jar -ORBInitialPort 1050 "C:\Users\Bruker\Downloads\train_triplets.txt\train_triplets.txt" 1  
Init finished
```

*Screenshot of server running*

## Cache:

For the implementation of the cache, we have created one interface, and two classes that implements that interface. Both classes has a HashMap which holds the different objects that are being saved in the cache.

The first cache-class is the user-cache. This cache can hold up to one thousand user profiles. If the cache is full, and you try to add another user profile to the cache, that new profile is only added if the user has played more total songs than one or more of the user profiles already in the cache. If this is the case, the user profile with the fewest amount of played songs is removed from the cache when this new user profile is added.

The second cache-class is the song-cache. This cache has no limit to the amount of songs it can hold, so there is no check executed when you try to add a song to it.

If cache is enabled on the server, the server has both a user-cache and a song-cache. To fill these caches with data, we have created an init method that goes through the database-file, and fills both caches with data. Since the database-file will stay the same through the lifetime of the server, there is no reason to do this more than the one time it is done at the startup of the server.

To fill the user-cache, we take advantage of the fact that all the lines of the database file are grouped based on the user ID. This means that we, on one read of the file, can go through all the user-song combination of one user, create a user profile, add the profile to the cache and then repeat this on the next user. By doing this, we end up with a user-cache containing the one thousand users that has played the most songs, without storing more than one thousand users in memory at all times.

The song-cache is a bit simpler, since we can store all the songs we find in the database file. This means that each time we get to a new line of the database file, we either add the song to the song-cache or update the information of the song in the song-cache, if that song already has been added to the cache.

On the client, we only use a user-cache, if cache is enabled. This cache starts of as empty, and gets filled up as the clients retrieves a user-profile from one of the methods it has called from the server. This means that only user-profiles already used by the client are stored in the cache. The same one thousand user limit also apply for this cache.