

CSCI398

Spring Session, 2013

Assignment 2: "Servant Management"

- [Aims](#)
- [Objectives](#)
- [Task](#)
- [Report](#)
- [Submission](#)

Aim

The aim of this assignment is for you to become acquainted with the mechanisms for managing CORBA servants (server side objects), and be semi-proficient at implementing client-server applications based on the CORBA model.

Objectives

The assignment involves implementation and correct deployment of a client-server system where the server process hosts "servants" that are from different classes and which have differing lifetimes.

On completion of this assignment, you should be able to:

- define server interfaces in CORBA IDL;
- work with the mappings of IDL to Java;
- generate client stubs and server skeletons from IDL;
- implement CORBA servants;
- construct POA based servers that use differing policies for different classes of objects existing in the same host process;
- utilize JDBC within a CORBA server;
- explain the concept of a "persistent CORBA reference"
- deploy a system with Sun's orbd CORBA daemon that is configured to restart a server process as needed.

You will find that the code is fairly similar to supplied examples. This is really a training exercise in which you become familiar with CORBA and its server side models.

Task

Just as a change from “student management packages” etc, your application is a “game server”.

This client server system will allow an authorized player to connect to the game server and start, or resume, play on a Sudoku puzzle. The server can generate puzzles with differing levels of difficulty; the player specifies the desired difficulty level when starting a new game. The server supports multiple concurrent players.

The client acts primarily as a display and interaction component; it shows the current state of a game, and accepts inputs from the user that place another digit in the puzzle. Each move made on the client is forwarded to the server. If a player wishes to suspend play, he/she simply closes the client application. The next time he/she connects, the puzzle state will be restored at the point of the last completed digit entry.

As each move is submitted, the server checks whether the puzzle is completely filled in – if it is complete then it is compared with a known solution and the server reports whether the user has won, or has made some error and has submitted an incorrectly filled in puzzle. (The server does not provide any feedback regarding incorrect guesses at the time they are made.) All the moves made by the user in their current game are recorded in persistent storage.

A player struggling to complete a game may realise that they have misplaced a digit – they can request that moves be undone, right back to the initial starting configuration. The server handles such a request by deleting the most recently entered data record, and returning details of the prior Sudoku puzzle configuration.

A player who has tired of an overly hard puzzle can request a new puzzle.

The server’s records for a player record how many puzzles they have completed, and how many they have abandoned, at each difficulty level. These status data are presented in a report shown when a player first connects and again after completion of a puzzle.

There is a second client program used by an administrator. This has limited options primarily relating to creation and deletion of password controlled player accounts.

The “administrator” component – instance of “AdminImpl extending AdminPOA” – is created with a persistent CORBA id. If the server process is not running, a request from the Admin client will be routed to ORBD and will result in the process restarting.

When the server process starts, CORBA objects (i.e. just IORs) are created for all players registered in the applications database and these are published as transient names in the name server part of ORBD. The server side player objects are managed with a POA that uses a servant manager – you are free to choose “default servant”, “activator”, or “locator” models. (I chose to use a locator approach with a small cache of player objects – this design requires re-accreditation with password on every request).

Illustration

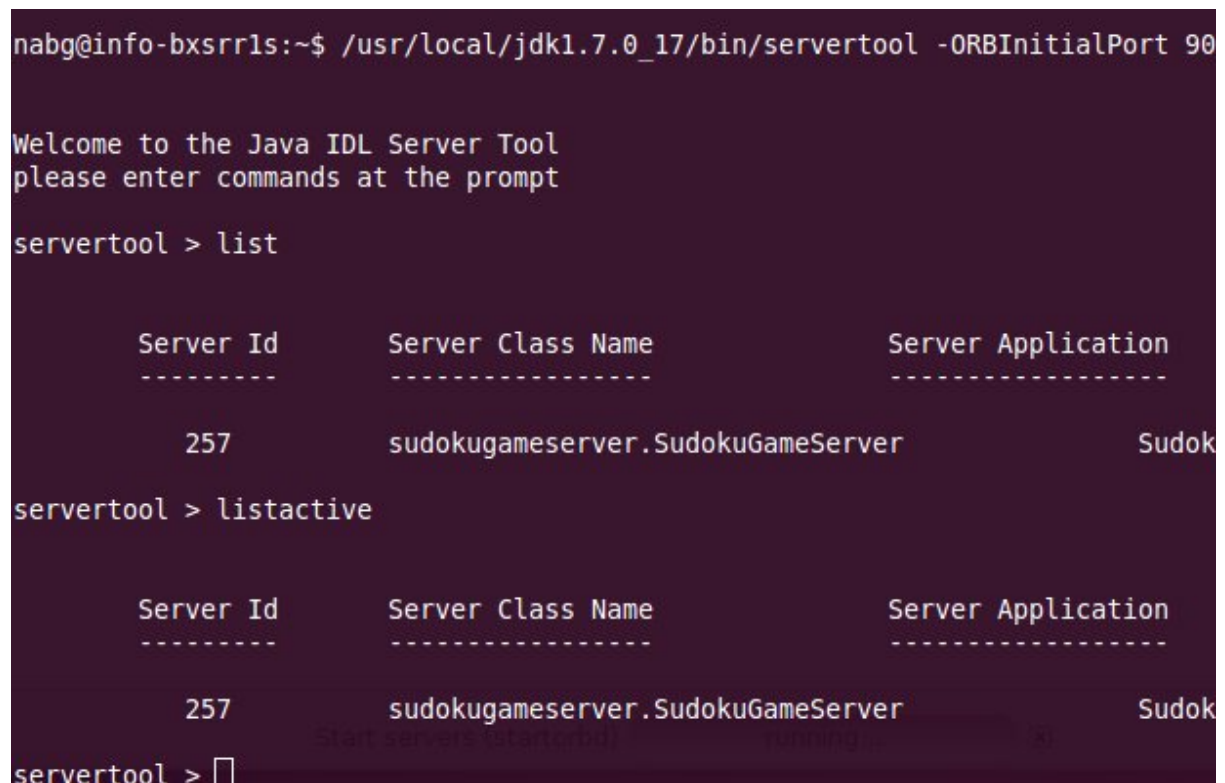
The following screen shots are from my version:

1. ORBD is started and then the server application is registered:



```
Output - Start servers (startorbd) 98-2013-Assignments/A2/SudokuGameServer/Servers.xml startorbd
startorbd:
Starting orbd at port 9090
```

The state of the server can be checked using the command line `servertool`:



```
nabg@info-bxsrrls:~$ /usr/local/jdk1.7.0_17/bin/servertool -ORBInitialPort 9090

Welcome to the Java IDL Server Tool
please enter commands at the prompt

servertool > list

      Server Id      Server Class Name      Server Application
      -
      257      sudokugameserver.SudokuGameServer      Sudok

servertool > listactive

      Server Id      Server Class Name      Server Application
      -
      257      sudokugameserver.SudokuGameServer      Sudok

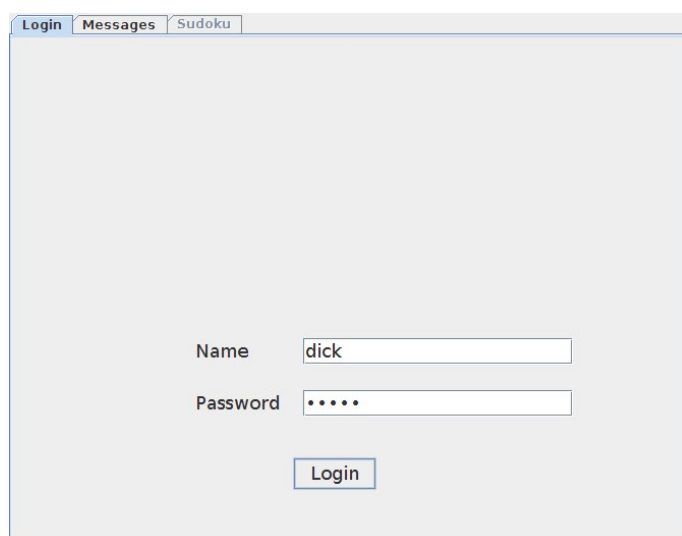
servertool > 
```

2. The admin client is a simple command line application – not particularly interesting. It has options for listing the names of registered players, creating a new player account, deleting an account, and for shutting down the server. Server shutdown involves two steps: removing the CORBA references for the player objects from the name server, and then `orb.shutdown`.

(References to players should be removed from the name service on shutdown. A player trying to connect will then get a “name not found” response rather than a low level CORBA exception that results from trying to use a connection to a process and port that are no longer there.)

```
Start servers (startorbd) x SudokuGameAdminClient (run) x
run:
Fri Aug 16 14:11:24 EST 2013
Established connection to admin component
You can list players, add players, delete players, or shutdown
list/add/delete/quit :list
Fri Aug 16 14:11:31 EST 2013
Listing players
dick
harry
jack
sal
sue
tom
list/add/delete/quit :add
Enter name and password
pete st0n3
Player record created
list/add/delete/quit :
```

3. The client has the typical “tabbed pane” interface for a Java client with “login”, “messages”, and Sudoku panels:



4. Incorrect player-name/password combinations result in a terse error message being displayed in the messages panel. A successful login will result in display of status data – has the player completed or abandoned any games and does he/she have a game currently in progress with its state saved on the server:

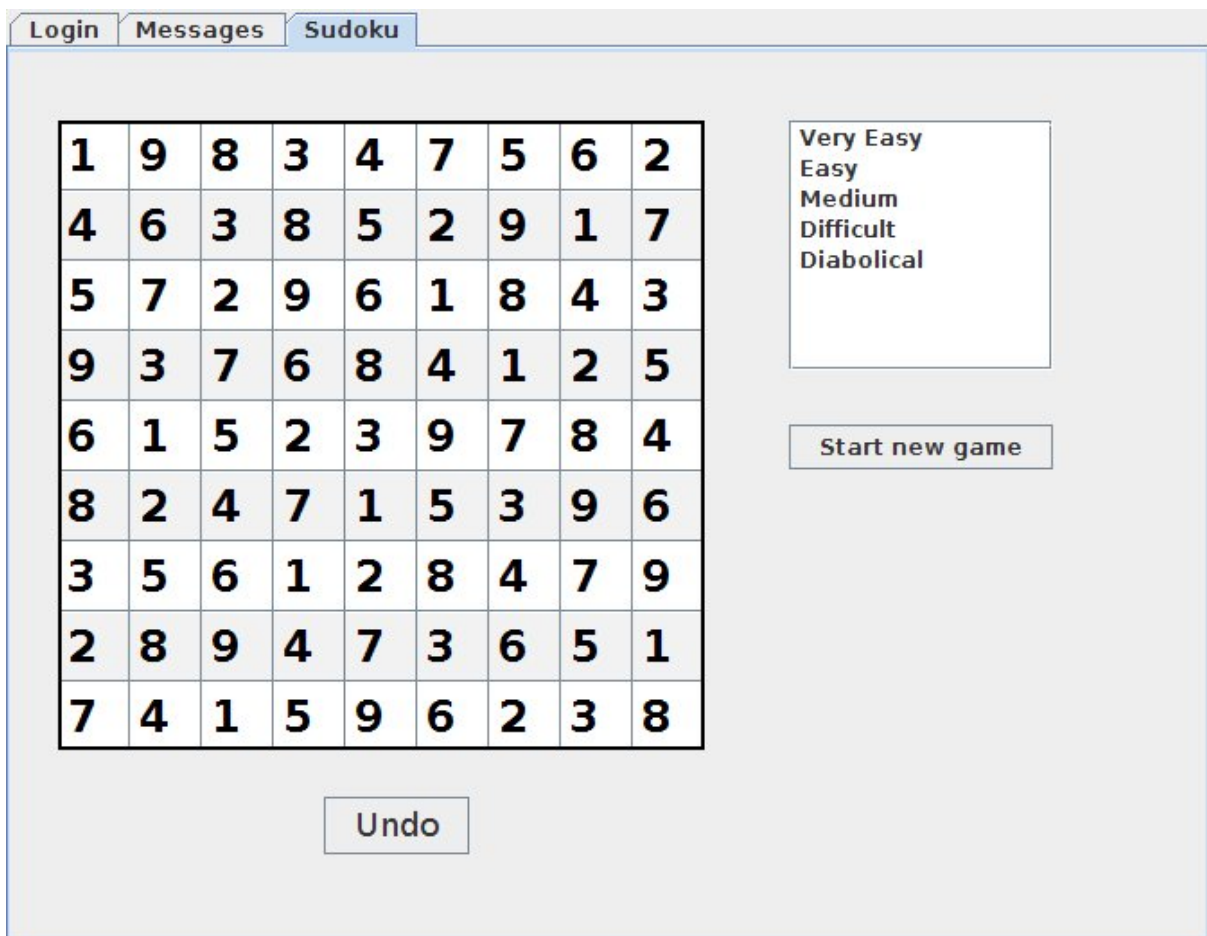


The report should show the number of games completed, and the number of games abandoned at each play level (very easy, easy, medium, difficult, diabolical). It should also state if there is a game in progress.

This example player has successfully completed two games – one at “very easy” and one at “easy” level. There is no game in progress.

5. The Sudoku panel shows the game and controls. If there is no saved game, it will show a completed configuration of an actual game (essentially, just a string constant).

If the player does have a game in progress, the display will show the state of the game as retrieved from the persistent storage managed by the server.



The controls consist of a list for selecting a game level, and “start new game” button, and an “Undo” button. (Game difficulty defaults to “very easy” if not chosen explicitly.)

Starting a new game, when there is a game in progress, causes that game to be “abandoned”. All data records relating to that game are deleted. The player’s record is updated (recording another abandoned game at a particular difficulty level).

The Undo control should be enabled when there is an existing game with some moves completed.

6. Client and server applications should use extensive logging so that you can debug your application. Some of these logs should be included in the evidence that you submit to show that your application does work:

```
run:
Trying to login as :dick;
Fri Aug 16 14:13:38 EST 2013
Connected ok with password, getting status details
Fri Aug 16 14:15:54 EST 2013
Requesting a new game
Got this game : 800000210003000000054700600000904000000080009081600020038005090000100030060040100
```

7. A new game will be displayed – with spaces for the values that are to be resolved:

login

Messages

Sudoku

8						2	1	
		3						
	5	4	7			6		
			9		4			
				8				9
	8	1	6				2	
	3	8			5		9	
			1				3	
	6			4		1		

Undo

Very Easy

Easy

Medium

Difficult

Diabolical

Start new game

(This is the game shown in the trace message).

8. You will design your own Sudoku game play code – or you may adapt code (with acknowledgement) from one of the many Java Sudoku games that you can find on the Internet.

I chose to use a table. Cells that contain values (either values that are part of the original game configuration, or values already entered by the user) are **non-editable**. Cells that display a space are editable. (By default, a JTable allows cells to be selected for editing by a double-click action, but you can configure your table to use single-click selection. An editing action only finishes when you click some other element in the table. I handle completion of editing by checking the input – looking for a single digit, anything else is discarded. The digit entered by the player is incorporated into the game configuration and a message is sent to the server.)

A first guess:

8						2	1	
		3				9		
	5	4	7			6		
			9		4			
				8				9
	8	1	6				2	
	3	8			5		9	
			1				3	
	6			4		1		

```
Have new config 800000210003000900054700600000904000000080009081600020038005090000100030060040100
Sending to server
```

9. I made a few more moves – incorrectly!

8						2	1	
	1	3				9		
	5	4	7	1	9	6		
	8		9		4			1
				8	1			9
9	8	1	6				2	
1	3	8			5		9	
			1	9			3	
	6	9		4		1		

(Two 8s in the same column, what was I thinking?)

This game system is meant to be similar to solving a puzzle in a newspaper. It doesn't provide any assistance; it doesn't highlight errors; it doesn't show you the choices for a particular square. (Such help features are common to other Java Sudoku games.) The only aspect that is an improvement on a printed paper game is the "undo" - if you make mistakes all your errors are cleaned away by undoing a sufficient number of times.

I undid a few moves getting back to:

8						2	1	
	1	3				9		
	5	4	7			6		
			9		4			1
				8				9
9	8	1	6				2	
1	3	8			5		9	
			1				3	
	6			4		1		

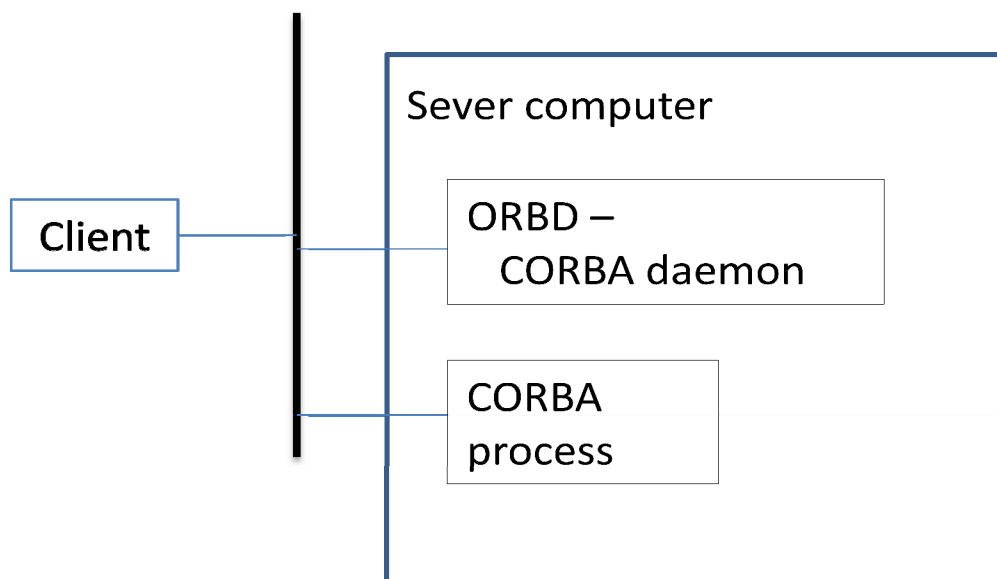
Undo

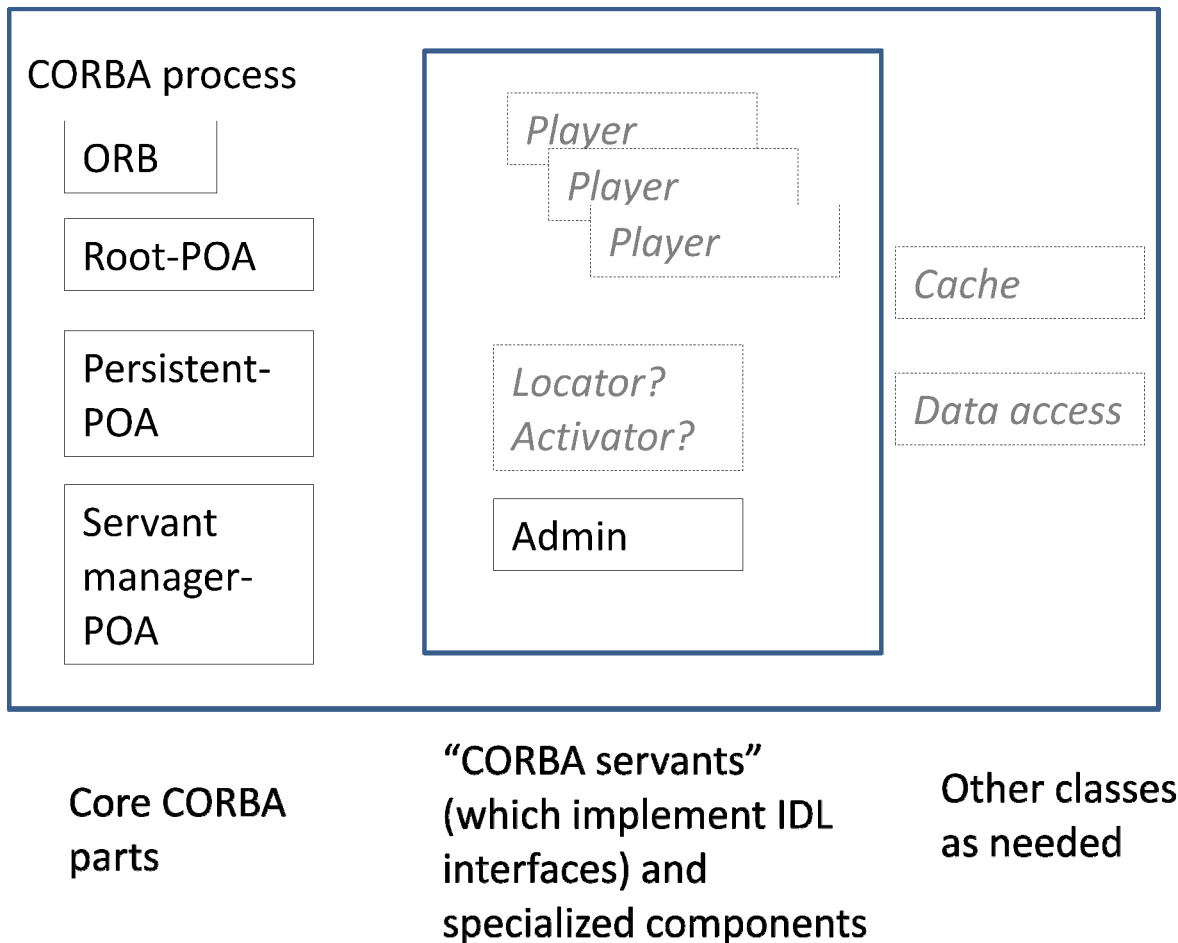
10, Example of a server side trace:

```
Binding name in nameservice  
SudokuGames/pete  
Binding object to name server  
Player locator preinvoke called for dick  
Room for another servant!  
Fri Aug 16 14:13:38 EST 2013  
Player object (re)created for dick  
Checking password  
Player locator postinvoke called for dick  
Player locator preinvoke called for dick  
Found servant dick
```

Architecture

The overall architecture is standard for a CORBA application:





Admin

- Create and delete player records in the database.
- List all players.
- Shutdown of server.

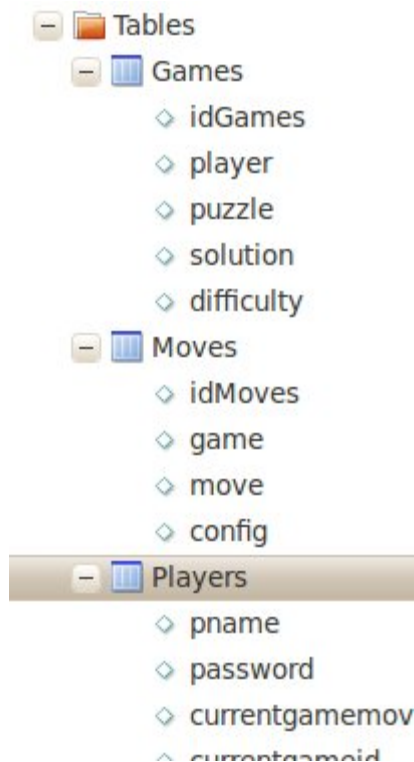
Player

- Check login.
(If you use a "default servant" or "locator" model you really need to reconfirm password on later requests as the server side object cannot maintain login status – it can be destroyed and later recreated).
- Assorted game options – your choice; it's things like `isGameInProgress()`, `getCurrentGame()`, `getStatus()`, `startNewGame(in short level)`, `recordmove()`, `undomove()`, ...

Data persistence

You will probably find JDBC more convenient than JPA for database access from the server process. JPA entity records cannot be used as CORBA data types so you would be copying data from the JPA entities into instances of classes generated from CORBA IDL struct

definitions. You might as well use the classes derived from the CORBA IDL throughout the application.



NetBeans projects

I had several projects:



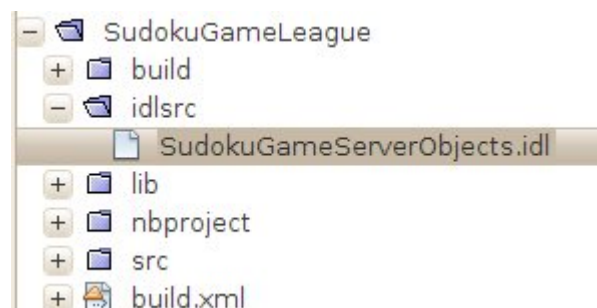
1. **SudokuPersistence:**

This was a “Java library” project – essentially just JDBC code to handle persistence in a MySQL database. I only use one database connection. The server will be multi-threaded so I make all the methods of my persistence management class “synchronized”.

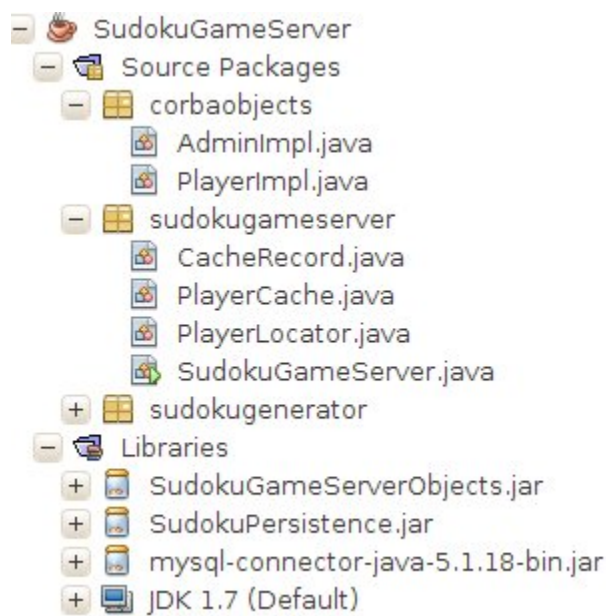


2. SudokuGameLeague

This was my IDL project where I define the CORBA interfaces for server side objects. It is a Java project with own ant file – but effectively it's another Java library project with the compiled .jar files needed in both server and clients.

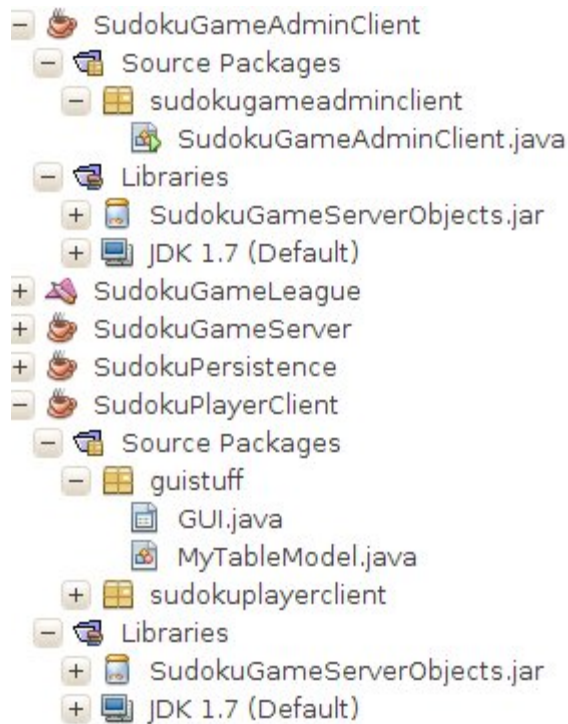


3. SudokuGameServer



There are three packages – corbaobjects for the servant classes, sudokugameserver for the mainline and the various helper classes needed, and sudokugenerator (code for generating puzzles with solutions at different levels of difficulty).

4. Clients



Sudoku code

You will find the code of the game generator in /share/cs-pub/398/A2. My code is derived from a Sudoku game on the Internet written by Eric Frankenberg (eric.frankenberg@iut-tlse3.fr) rewritten for clarity (and increased speed though that isn't important). You can download his original code – he has a GUI game play component, you might wish to base your GUI code on his. (There are several other Java Sudoku games with GUI interfaces that you might prefer to use.)

Report

Your assessment will be based on a report (A2.pdf) that you compose. This report should be submitted as a PDF file.

Your report should have

- A context section (800+ words) where you explain in your own words (not words cut and pasted from either me, or the OMG) the role of "object adapters" and the daemon processes that are used in this application. You might find it worth including some diagrams illustrating the deployment of the processes and showing how a client's action might cause a server process to be started and objects to be instantiated within that process.

- An IDL section that includes your definition of a module with the interfaces for the “Admin”, and “Player”. Your IDL may also have to define a number of simple structs and sequences.
- Server-side code:
You should use separate subsections to present the code, and any supplementary commentary on your code, for:
 - The main server process itself along with the code that sets up POA objects and creates persistent objects.
 - The Admin implementation class
 - The Player implementation class
- Client code
- Evidence for correct operation:
This should include screen-shots and some tracers. Include System.out tracer statements in client and server code. The tracers should note operations that are performed. (Tracer output will be found in files created in subdirectories of the "orbd" directory that orbd will create for the project.) It would probably be advantageous if you presented the tracer outputs in two column text and show the relations between client and server-side actions. You should highlight things like objects being deactivated. (You can add “finalize” functions that show when they get garbage collected – but you will only see that happen if you add server side code to force garbage collection at regular intervals.)

Where you have used my code suppress details in any listings, e.g.:

```
private static void registerObjWithNameService(NamingContext root,
        NameComponent[] serverName, org.omg.CORBA.Object obj) throws
        InvalidName, AlreadyBound, CannotProceed, NotFound {
// As standard
...
}
```

Do NOT include the code of any file generated by the IDL compilers.

Screen shots are easy Ubuntu - the "Applications/Accessories" menu has a screenshot tool.

Submission

The due date for submission will be announced in lectures; the assignment is provisionally set for Friday September 13th.

The assignment is to be submitted using "turnin"; you must be logged in on banshee to use turnin, turnin does not send you email and gives only a terse acknowledgment.

Overall presentation of report (1)

Explanation of role of object adapters, daemons etc (1)

IDL (1)

Clients (3)

Server and persistence (4)