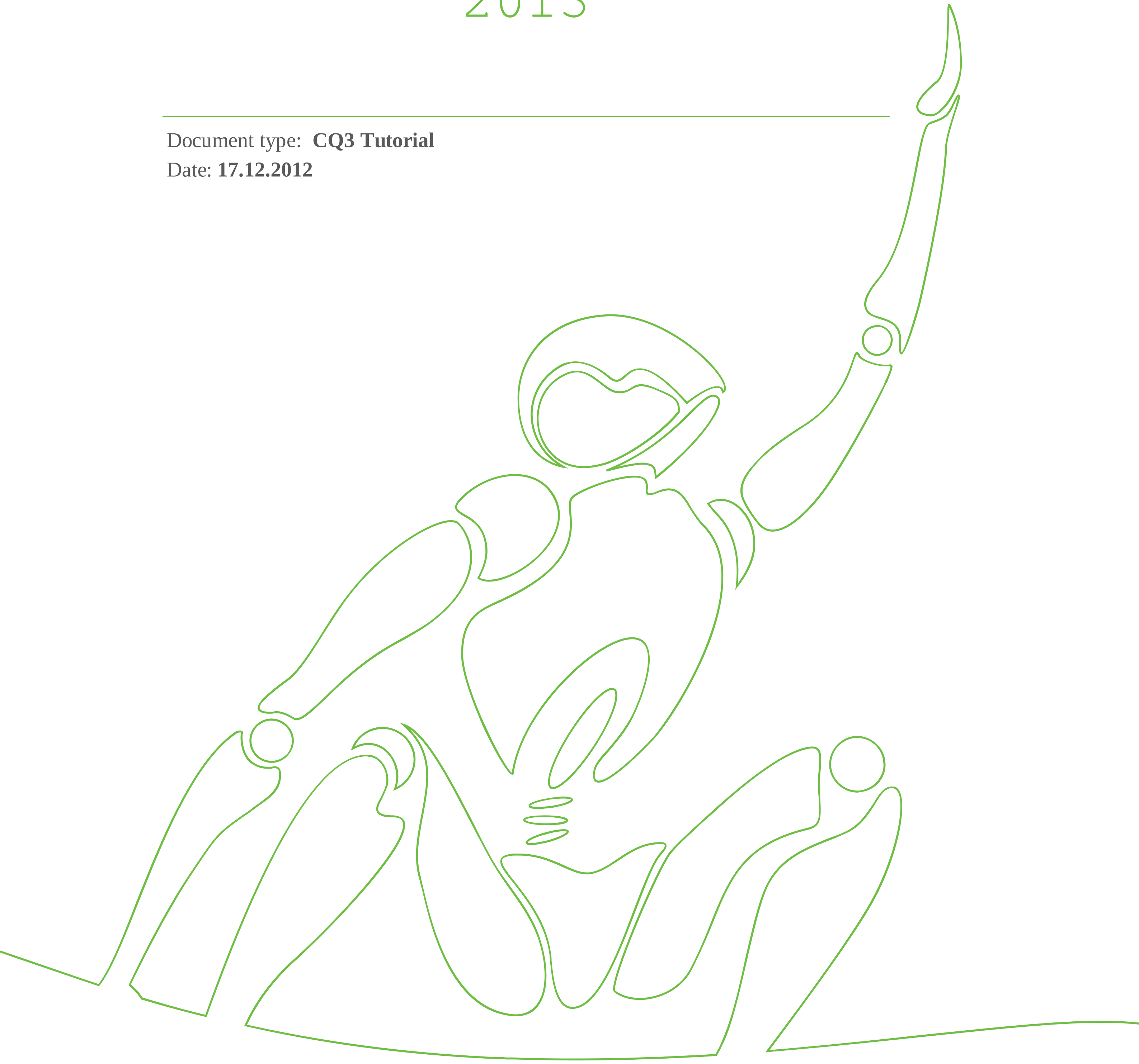
AI-MAS Winter Olympics 2013

Document type: **CQ3 Tutorial**

Date: **17.12.2012**



Crafting Quest 3 - Tutorial

Introduction

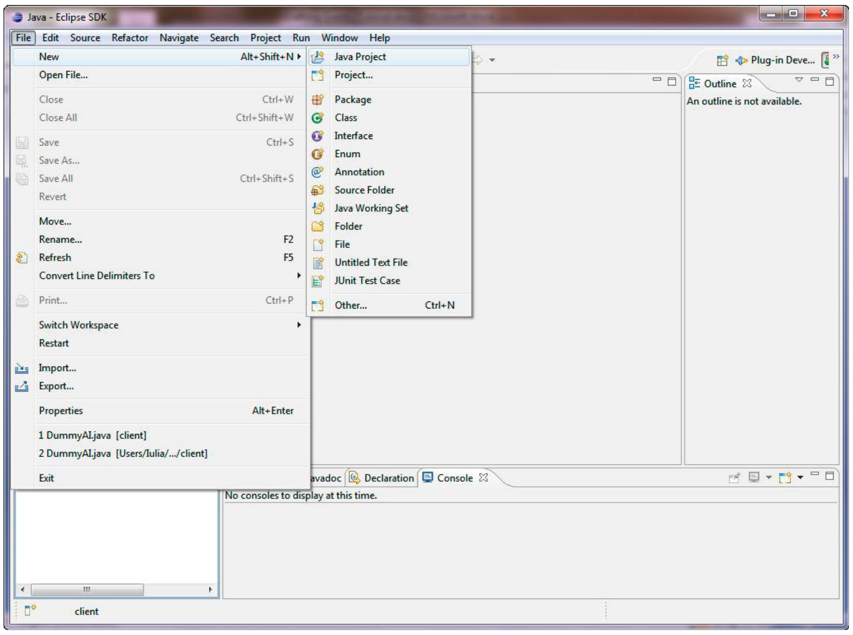
Crafting Quest is a turn-based strategy game with maximum 4 players. This tutorial describes how to run both the client and server software. It also shows how to build the jar archive that you will have to submit for the competition. This tutorial is based on the Eclipse IDE.

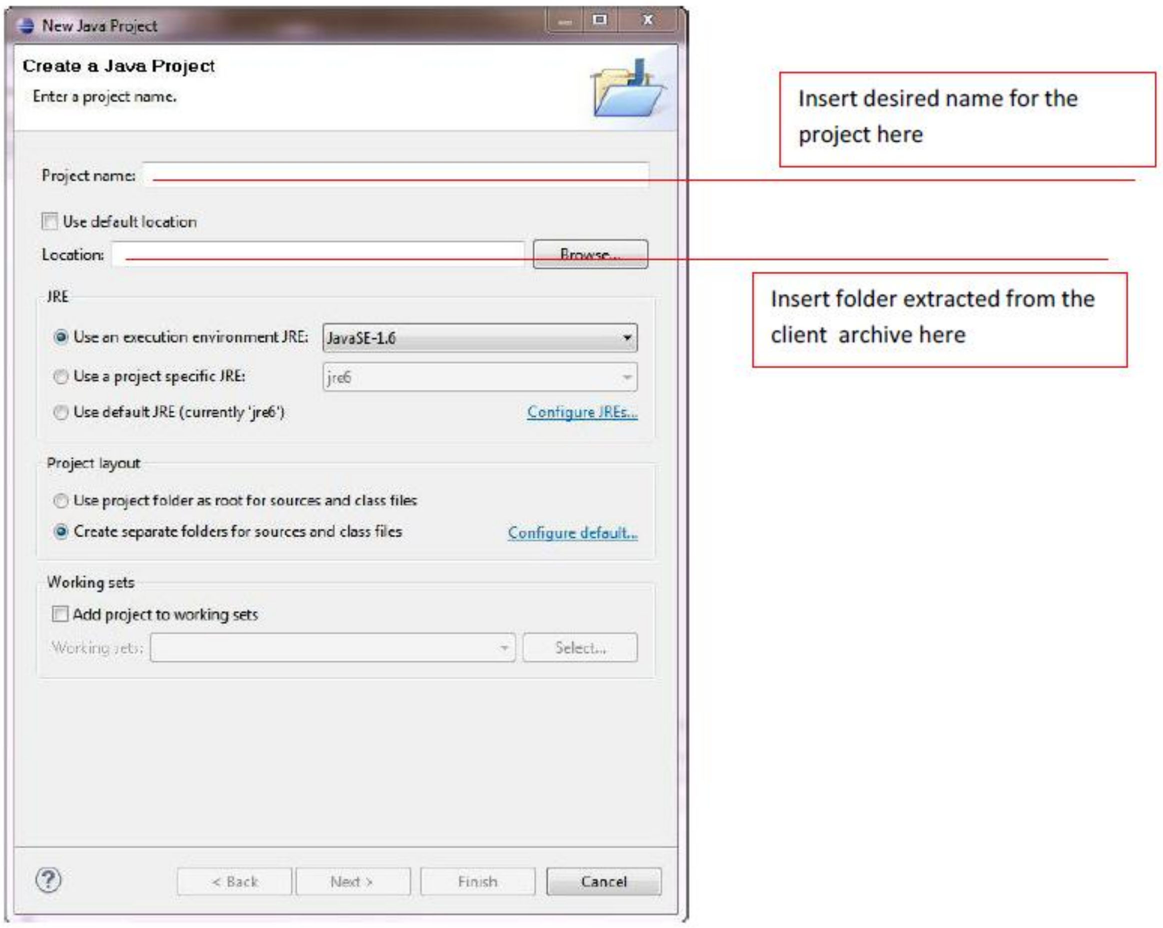
Preconditions

The following assumes that you already have your Eclipse working environment set up.

Running the client

In order to run your solution, you should create a new Java project, setting the folder extracted from the cqclient.zip archive as default location (uncheck *Use default location* and set the *Location* field) Set your desired name for the project in the *Project Name* field and press *Finish*.




New Java Project

Create a Java Project

Enter a project name.

Project name:

☐ Use default location

Location: Browse...

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre6

( ) Use default JRE (currently 'jre6') Configure JREs...

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files Configure default...

Working sets

☐ Add project to working sets

Working sets: Select...

< Back Next > Finish Cancel

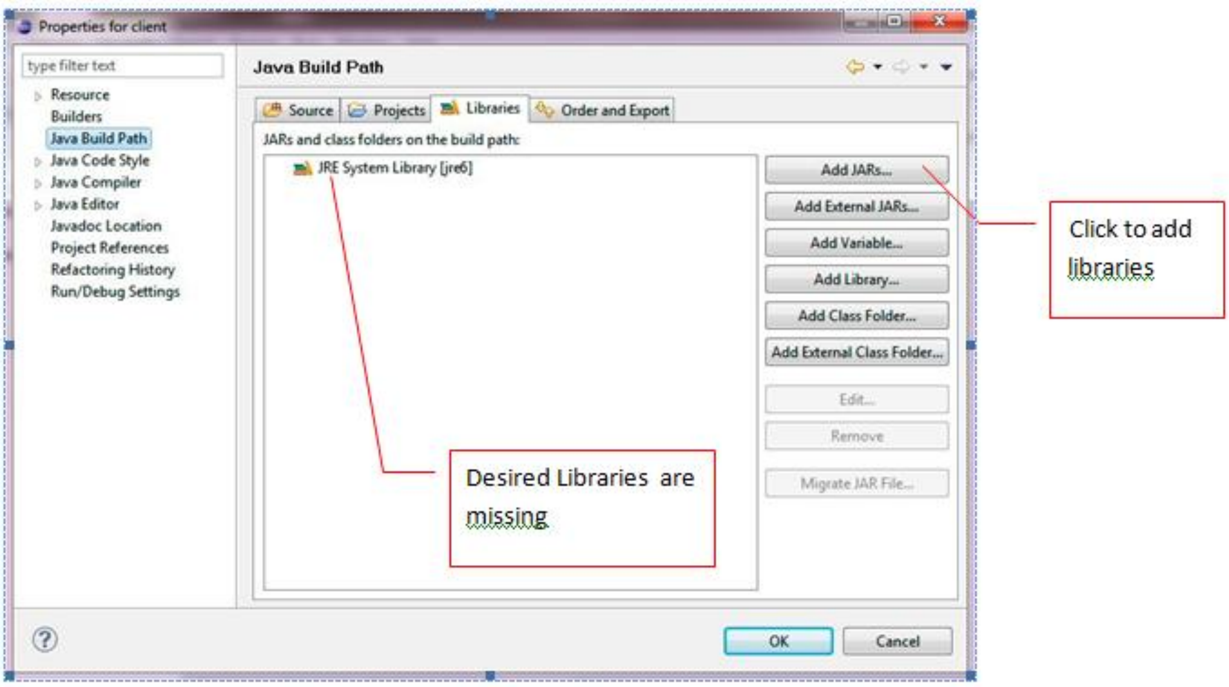
Insert desired name for the project here

Insert folder extracted from the client archive here

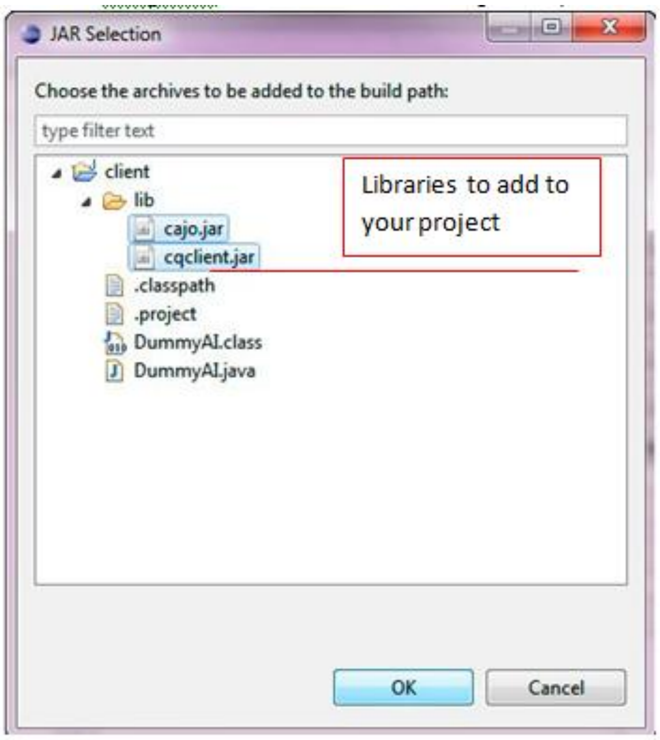
If the following libraries:

- cajo.jar
- cqclient.jar

do not appear in the project (select project, right mouse click and select *properties*, then go to the *Java Build Path* tab and look in the list for them).



Otherwise, select your project, right mouse click and select *properties*. In the *Java Build Path* tab, under libraries select *Add JARs* and choose the three libraries from the *lib* folder. Hit *Ok* twice to complete(*Jar Selection* and *Properties*).

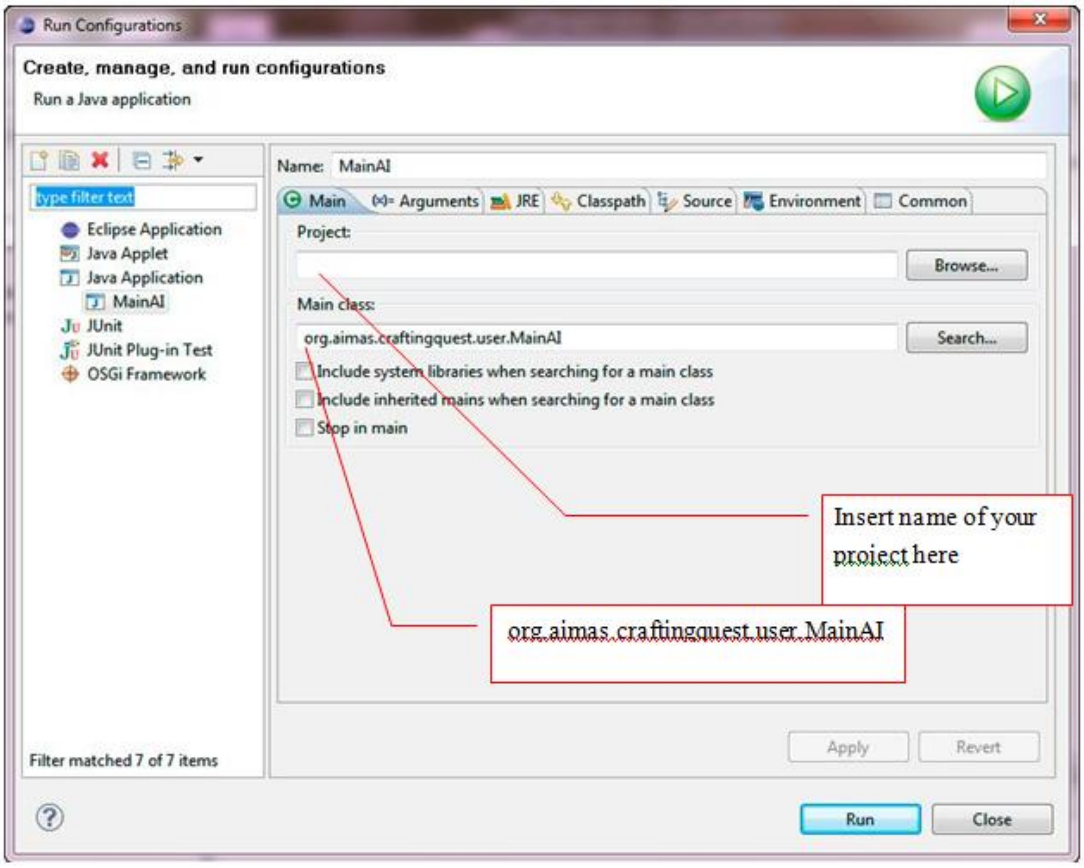


NOTE: The client SDK ships with the **javadoc** documentation for the client API. It is contained in the **cqclient/doc** folder that you can access after unzipping **cqclient.zip.** To enable documentation for the client API in Eclipse, follow the steps in this short tutorial: http://www.cs.laurentian.ca/aaron/cosc1047/eclipse-tutorials/javadoc-tutorial.html#javadoc2, where the target jar file is **client.jar** and the javadoc folder is **cqclient/doc.**

In order to run the client (as an Application), you should set a RunConfiguration for your project (Run → Run Configurations or right click on project and select Run → Run Configurations)

It should have the following settings:

- org.aimas.craftingquest.user.MainAI must be set as Main class
- Insert these arguments:
    - ClassName – name of the class containing your solution
    - Host – probably localhost for your client
    - Port – must be identical to the server's port
    - ServerName – the name set for the server as a run argument (See Running the Server)
    - ServerKey – a secret key defined in the secrets file from the server's archive


Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

Eclipse Application
Java Applet
Java Application
MainAI
JUnit
JUnit Plug-in Test
OSGi Framework

Filter matched 7 of 7 items

Name: MainAI

Main | Arguments | JRE | Classpath | Source | Environment | Common

Project:

Main class:

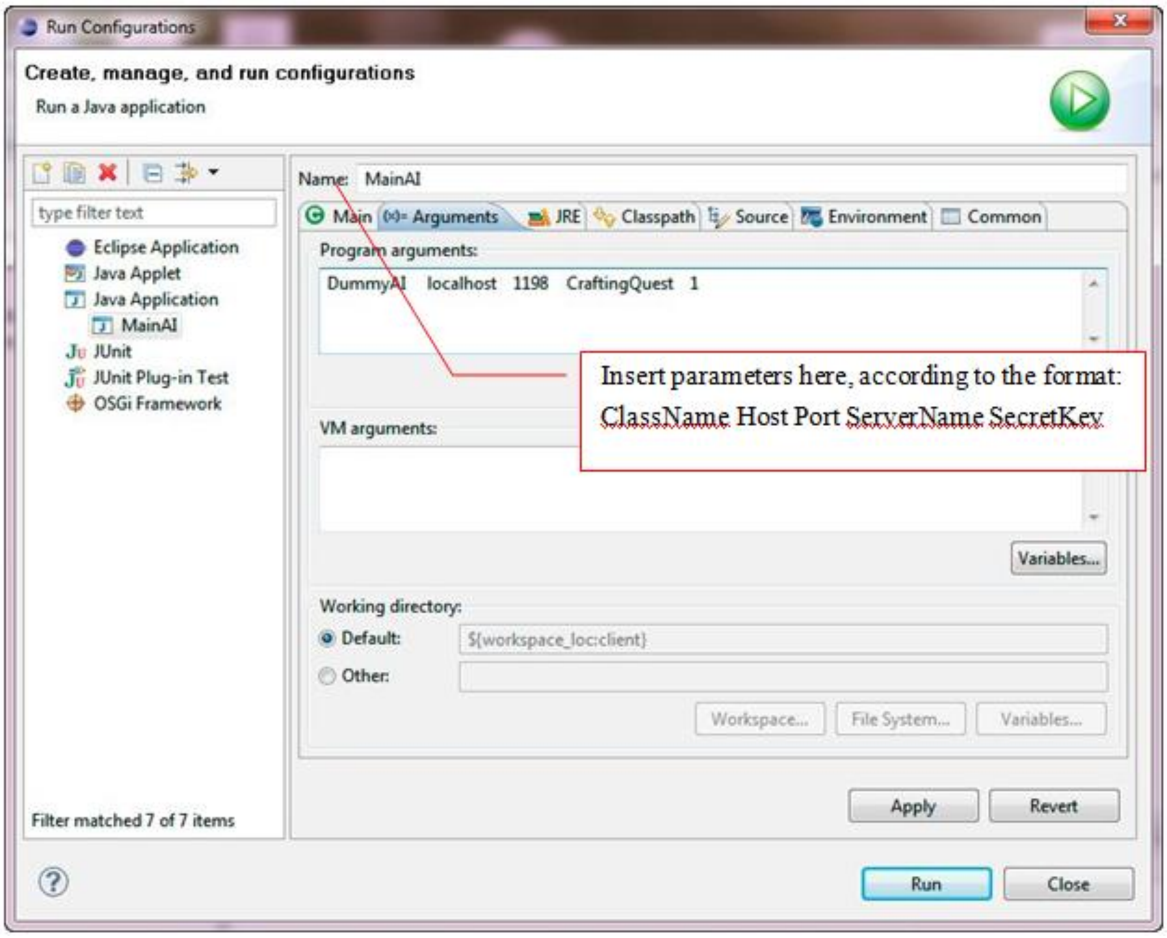
org.aimas.craftingquest.user.MainAI

Include system libraries when searching for a main class
Include inherited mains when searching for a main class
Stop in main

Insert name of your project here

org.aimas.craftingquest.user.MainAI

The image below contains the arguments for running our Dummy solution. During the competition, those parameters will be set automatically by scripts responsible for launching the game.


Run Configurations

Create, manage, and run configurations

Run a Java application

Name: MainAI

Program arguments:

```
DummyAI localhost 1198 CraftingQuest 1
```

Insert parameters here, according to the format:
ClassName Host Port ServerName SecretKey

Working directory:
Default: ${workspace_loc:client}

Click *Run* only after you have unzipped and started the server.

Running the Server

To run the server use the following command:

```
java -jar cqserver.jar [ServerName, Port, Secrets_file_name]
```

having the parameters defined above. The parameters in square brackets are optional. Default values exist for them.

The default values for these parameters are:

- **serverName**: *CraftingQuest*
- **Port:** 1198
- **Secrets_file_name:***secrets.txt*

If you do choose to change the values for the server parameters you have to specify all three of them, otherwise the server will signal an error.

If you want to enable the **GUI** you have to add an additional parameter to the above command, namely:

java –Dgui=true -jar cqserver.jar [ServerName, Port, Secrets_file_name]

Secrets.txt is a text file having the following format:

- The first line contains an integer ***n*** specifying the number of clients that must connect to the server
- The next ***n*** lines each contain an integer specifying a secret key

The secret keys (integers) are used for identification and security reasons and will be automatically generated by scripts running a game during the competition.

NOTE: The game configuration shipped with the SDK (specified in the **GamePolicy.xml** file), provides for a number of **2 players** on the map. If you wish to modify this setting to include up to **4 players** (the maximum allowed) you can edit the **GamePolicy.xml** file and alter the parameter *<nrPlayers>* to the value you desire. **Please make sure** to reflect the change in the file **Secrets.txt** as well, by defining **n** secret keys (where **n** is the number of players you have set in **GamePolicy.xml**).

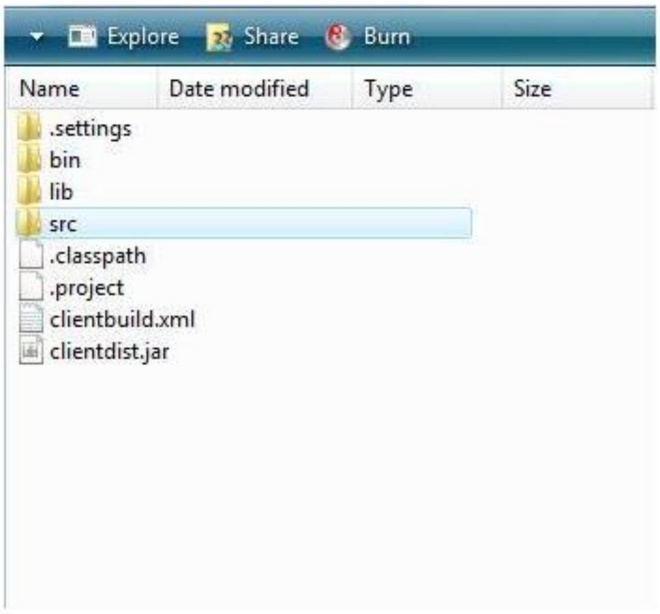
Setting custom logging configurations

Both client and server archives contain a file named **logging.properties** which is a **log4j** properties file, i.e. it gives directives for the way in which messages are logged. The default settings generate a *ConsoleAppender* which will print messages to *System.out*. By un-commenting the rest of the directives you can also have the logging mechanism output messages to files called server.log and client.log respectively. Additionally, you can customize the **logging.properties** file in any way you like to obtain desirable logging configurations. For details about this visit http://logging.apache.org/log4j/1.2/manual.html.

Building the solution needed for submission

Building your solution is done by running the **ant build script (clientbuild.xml)** included with the client archive (cqclient.zip). Running the script requires that you have the **ant** utility installed on your system. You can download it from **http://ant.apache.org/bindownload.cgi** and set it up**.**

Let's assume that your project directory looks something like this:



Name	Date modified	Type	Size
.settings			
bin			
lib			
src			
.classpath			
.project			
clientbuild.xml			
clientdist.jar			

The build script assumes that all your source files lie in the **src** directory. You must ensure this in order for the script to function properly. If these conditions are met, navigate to the root of your project (where **clientbuild.xml** resides) and run the following command:

```
ant  -Dmainclass=<your_main_class>  -buildfile clientbuild.xml
```

The output of this command will be a jar archive named **clientdist.jar** which you will have to submit as your solution for the competition. The parameter **<your_main_class>** represents the **fully-qualified java class name** of your solutions' main class (the one that **extends AIThread**).

Note: A fully-qualified class name is the name of a java class that includes its package name. Running the above command will also produce a **build** directory in the project root. After building your jar, you can delete the build directory.