1. Design Documentation:
   1. A complete UML document of the game package is located in the doc subdirectory of the packaged source code.
   2. The title of the UML document is: EricSabelhausSorcerersCaveUML.png
2. User's Guide: description of how to set up and run your application
   1. Setup of this application requires at least Java version 1.7
   2. In order to run the application, the following command must be run within the bin directory of the compiled application code:
      1. java -classpath bin game.GameGUI
   3. Once the application is running, select one of the three available caves to explore. Listed below:
      1. SmallSimpleCave.txt
      2. SmallCaveWithUndiscoveredItems.txt
      3. LargeCave.txt
   4. After you’ve selected your cave, you will be presented the sorcerers cave GUI
   5. Going from top to bottom, the first element you see is the search bar, which is followed by the cave navigation tabs.
      1. Search Bar:
         1. To the left is the submit button
         2. In the center is the input field to enter a search string
         3. To the right is a selection panel where you may choose to search by Name, Type, or by Index.
      2. Navigation Tabs
         1. Going from left to right, the parties within the cave are accessible via their tab
         2. In each tab, the party selected will be displayed in a space delimited text format
   6. Searching:
      1. When a search is executed, the content of the item searched for will be displayed in a popup window. If no item is found, or an illegal search parameter is entered, the appropriate message will be returned instead.
3. Test Plan: sample input and ***expected*** results, and including test data and results, with screen snapshots of some of your test cases
   1. Sample Input
      1. The following is the sample input provided during testing:

// p:<index>:<name>

p : 10001 : Unity

// c:<index>:<type>:<name>:<party>:<empathy>:<fear>:<carrying capacity>

c : 20001 : Woman : Lucy :10001 : 17 : 22 : 20

c : 20002 : Woman : Jane :10001 : 22 : 15 : 25

// t:<index>:<type>:<creature>:<weight>:<value>

t : 30001 : Gold : 20001 : 50 : 2000

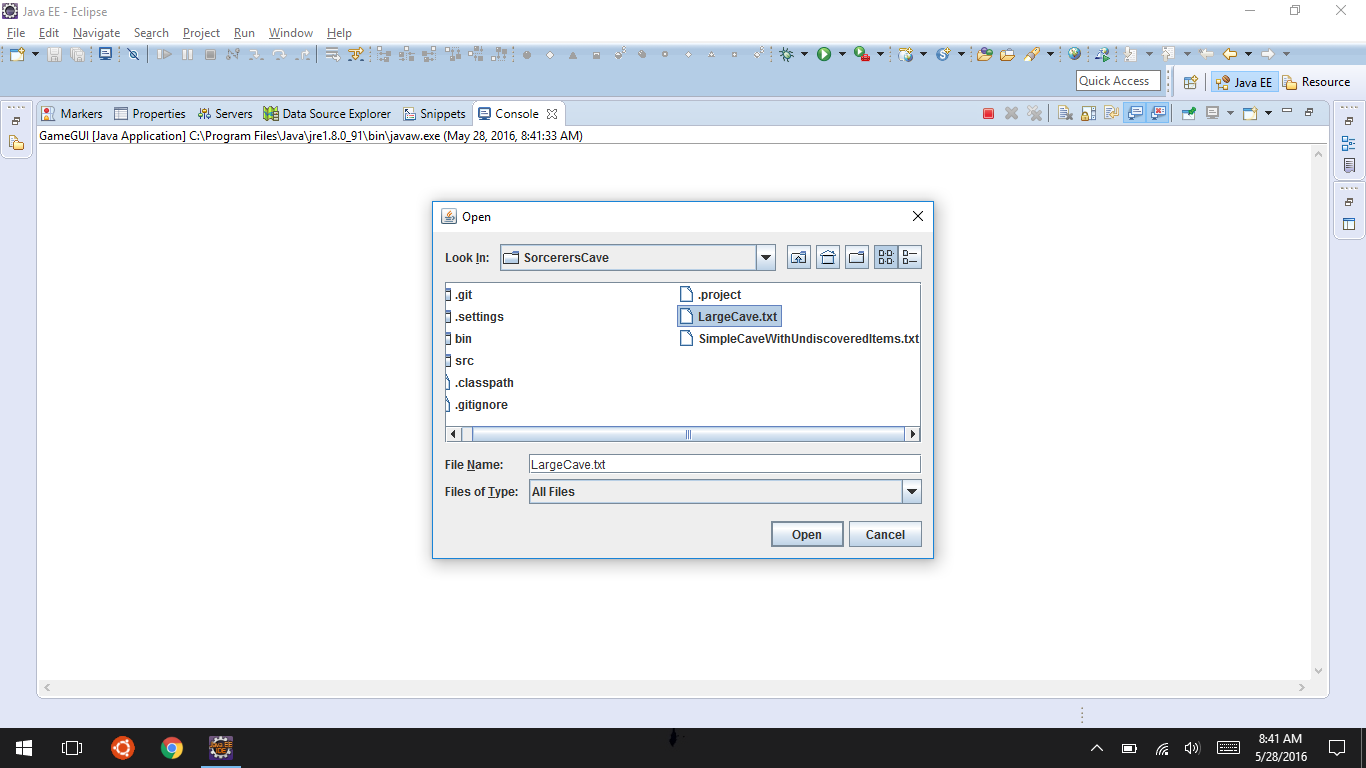
t : 30002 : Gold : 0 : 75 : 5000

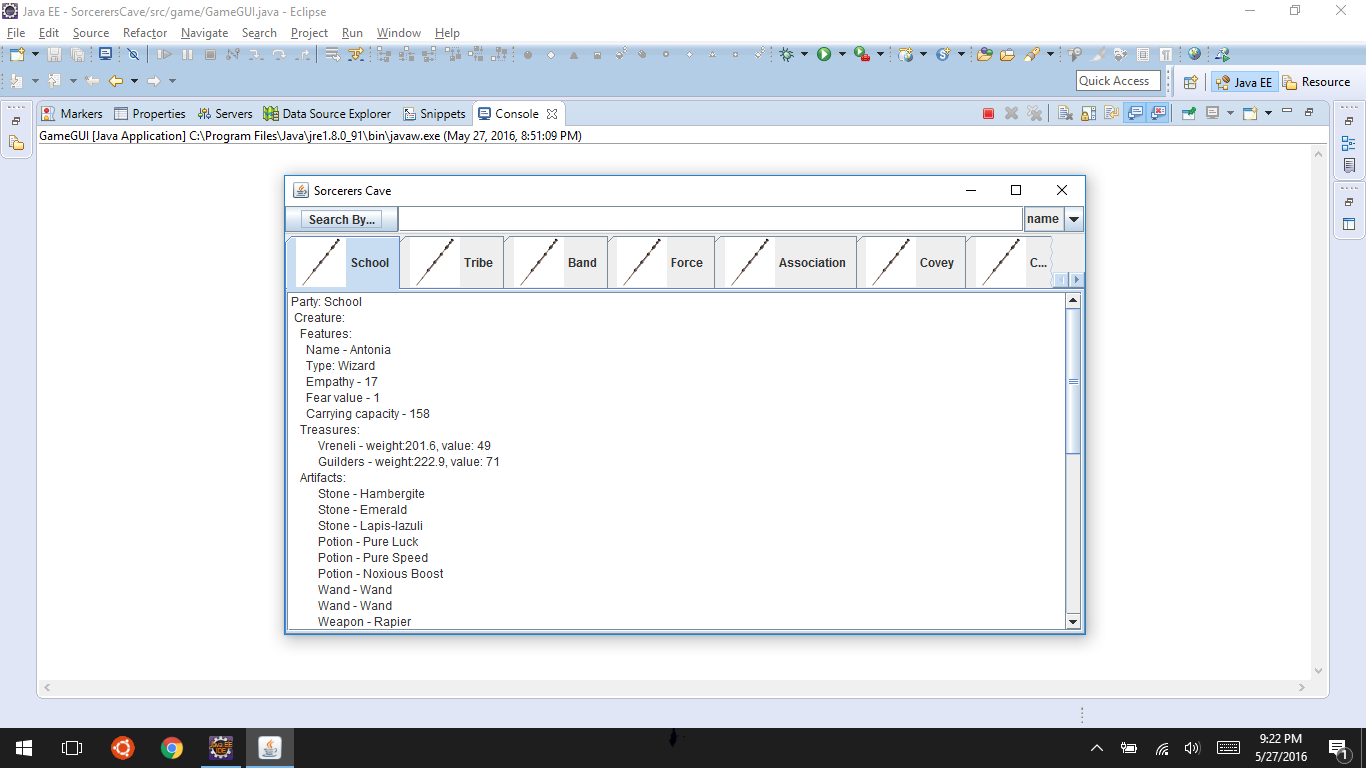
t : 30003 : Gems : 20002 : 10 : 10000

// a:<index>:<type>:<creature>[:<name>]

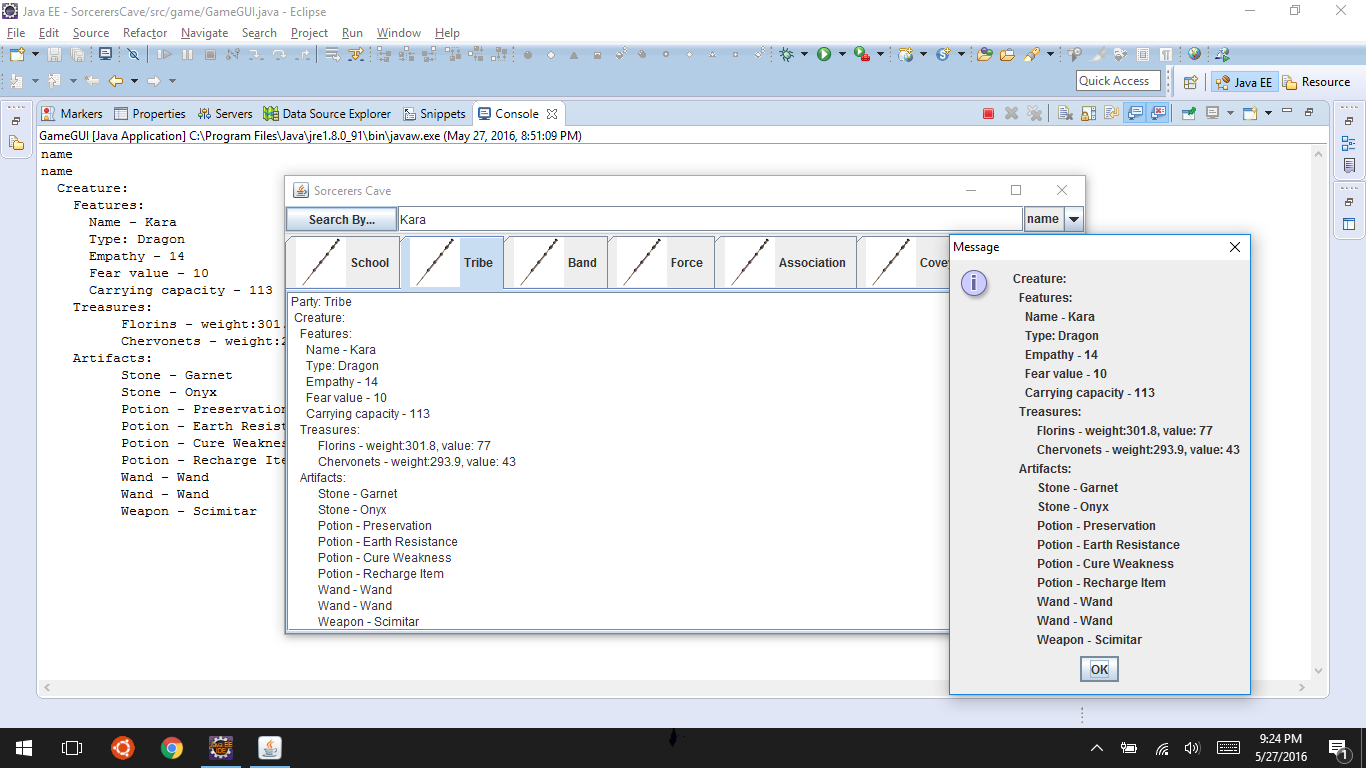
a : 40001 : Wand : 20001 : ElderWand

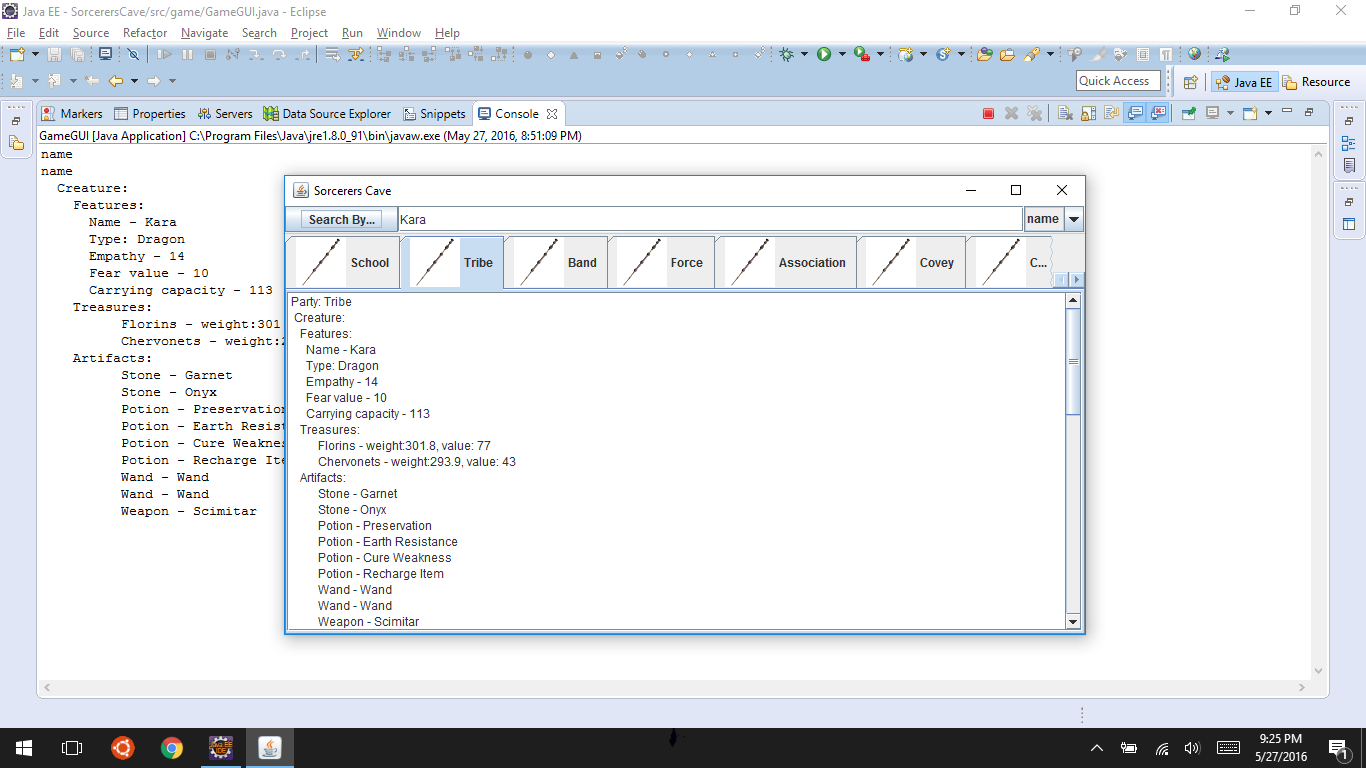
* + 1. The expectations are as follows:
       1. One Cave is created
       2. One Party is created
          1. Must belong to the one cave
       3. Two Creatures are created
          1. Both must belong to the one party
       4. Three Treasures are created
          1. One owned be each creature
          2. One must be undiscovered
       5. One Artifact is created
          1. Must belong to the first Creature
  1. Result
     1. Assertion tests using Java JUnit testing framework succeeded in 0.130 seconds
  2. GUI Test Cases:
     1. Picking LargeCave.txt using JFilePicker. File contains no undiscovered items. After file loaded, all parties are present in tabs with no undiscovered items present as expected



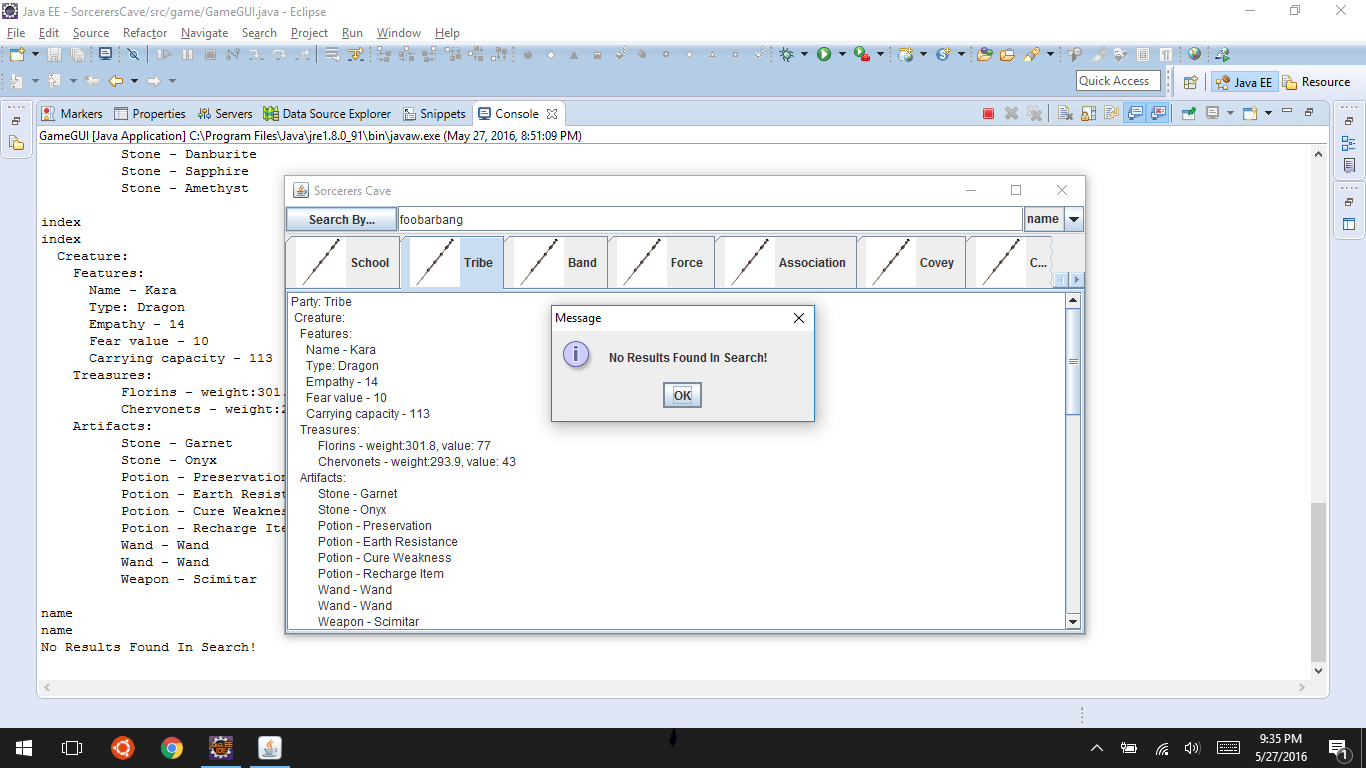


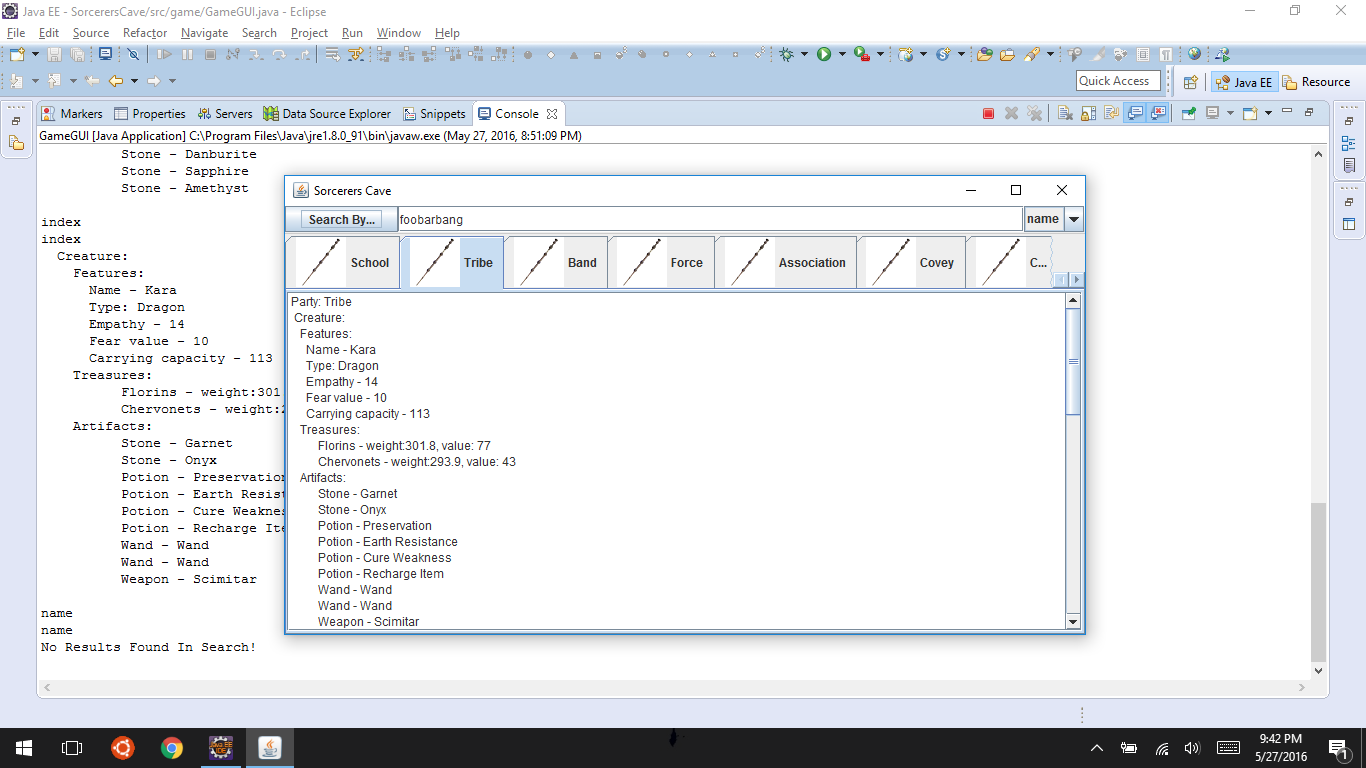
* + 1. Search by name for character named Kara returns expected creature. Pressing OK button after response given returns focus to GUI as expected



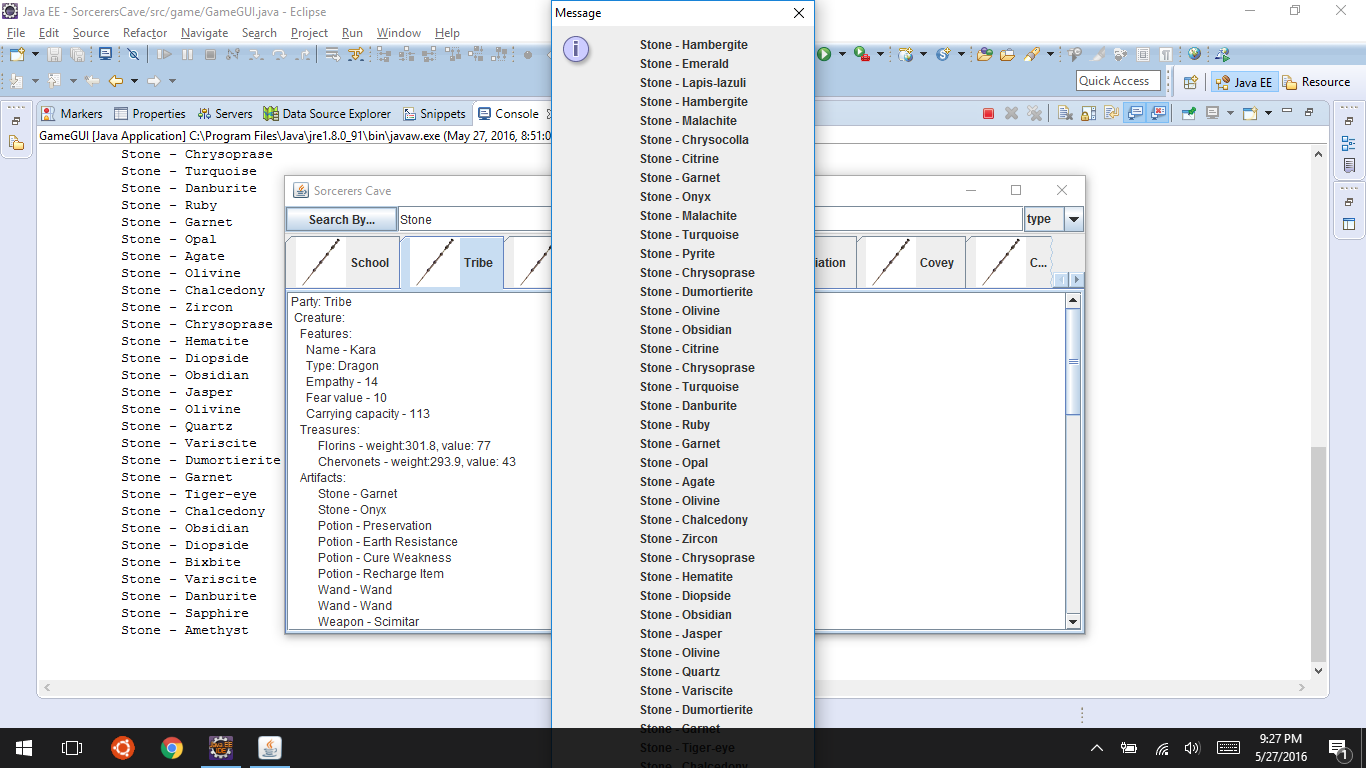


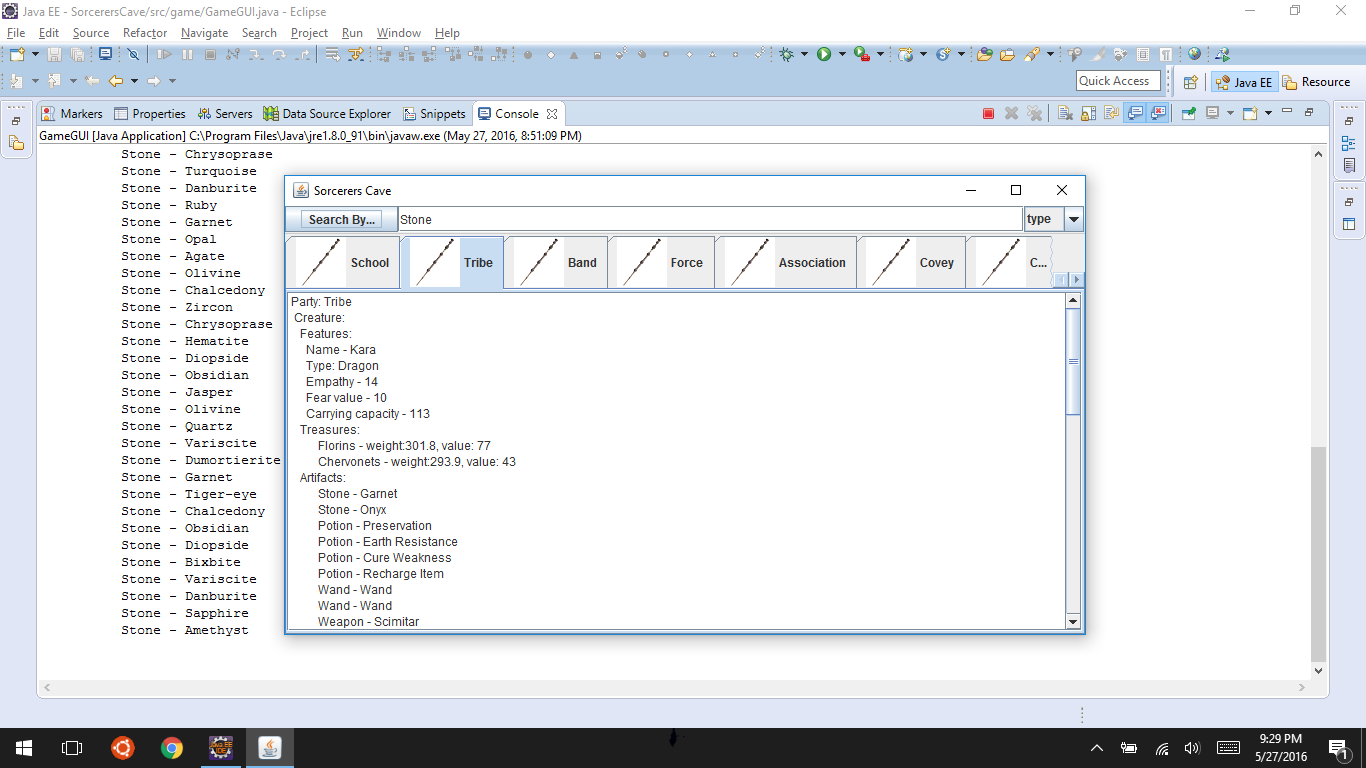
* + 1. Search by Name for foobarbang returns expected standard output. Pressing OK button after response given returns focus to GUI as expected.



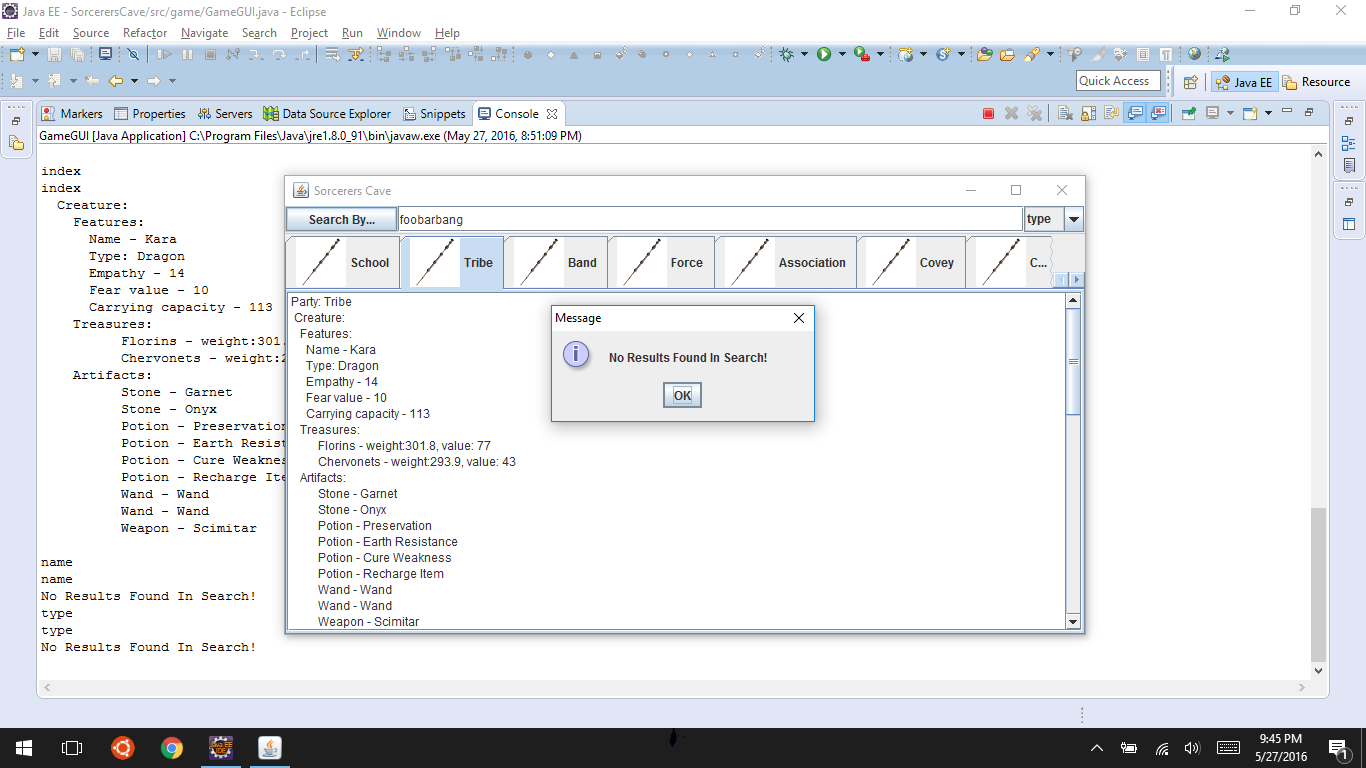


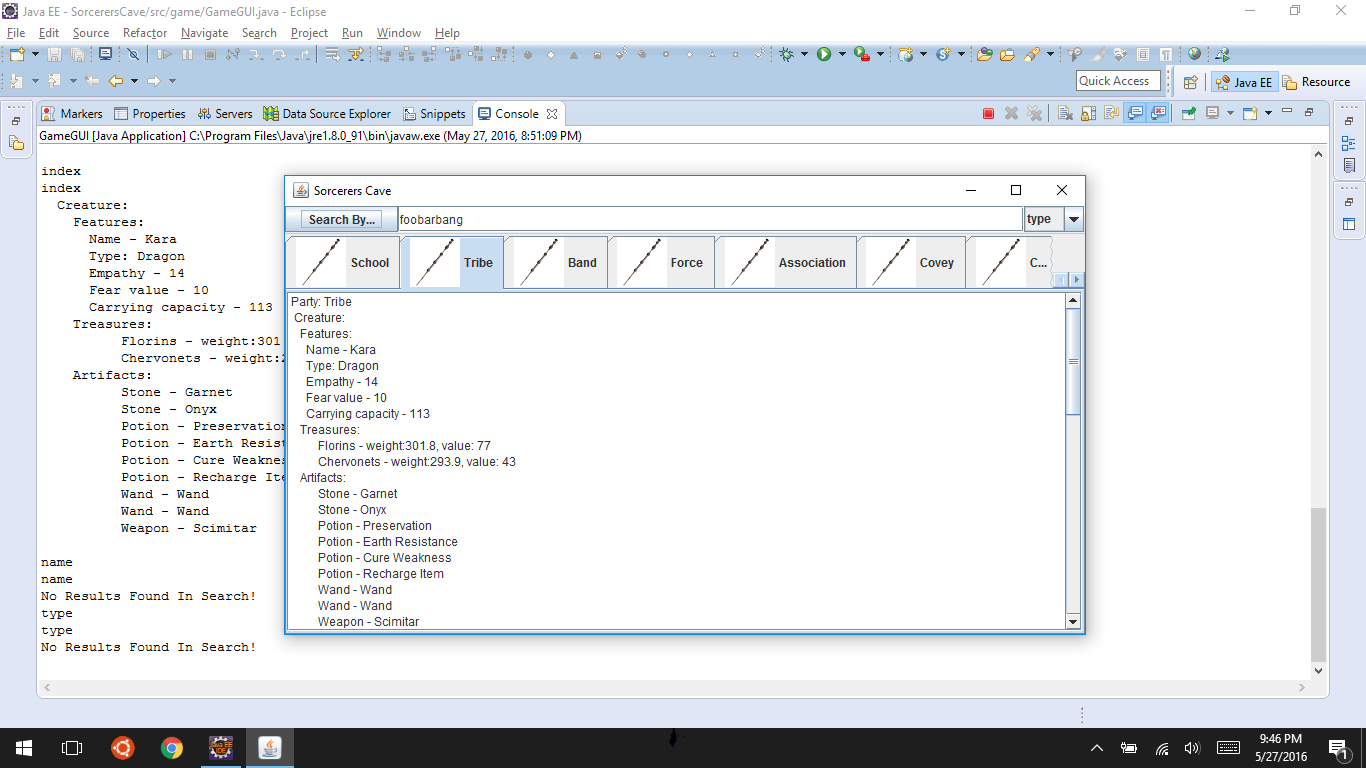
* + 1. Search by type of Stone returns all Stones as expected. Pressing OK button after response given returns focus to GUI as expected.



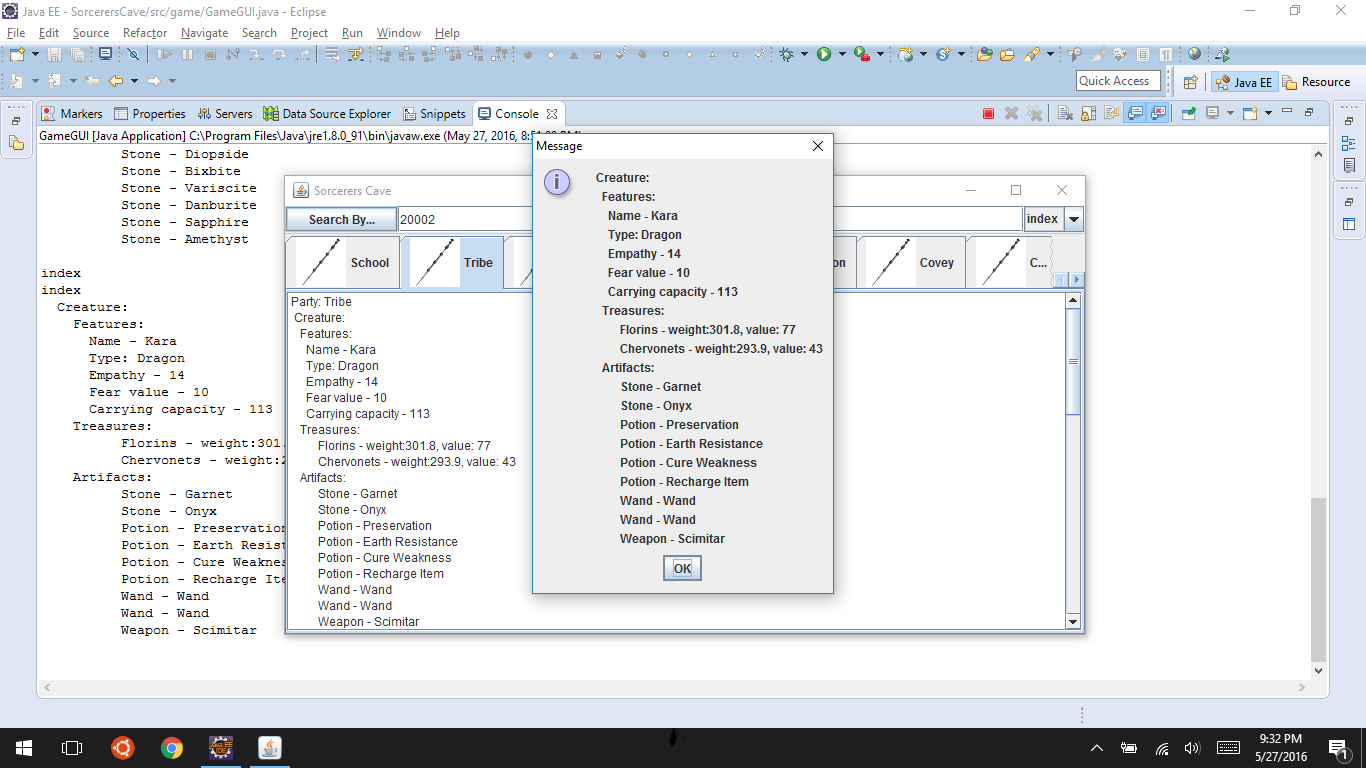


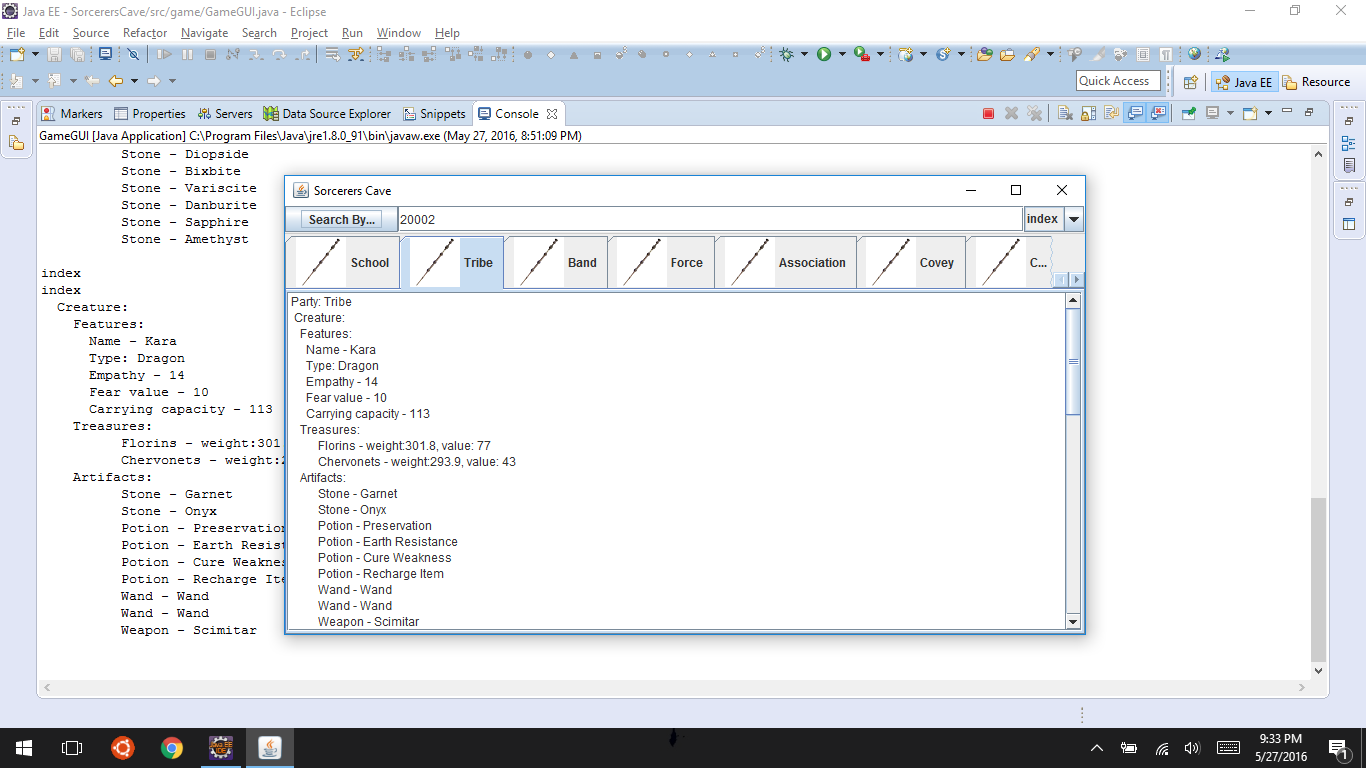
* + 1. Search by Type for foobarbang returns expected standard response. Pressing OK button after response given returns focus to GUI as expected.



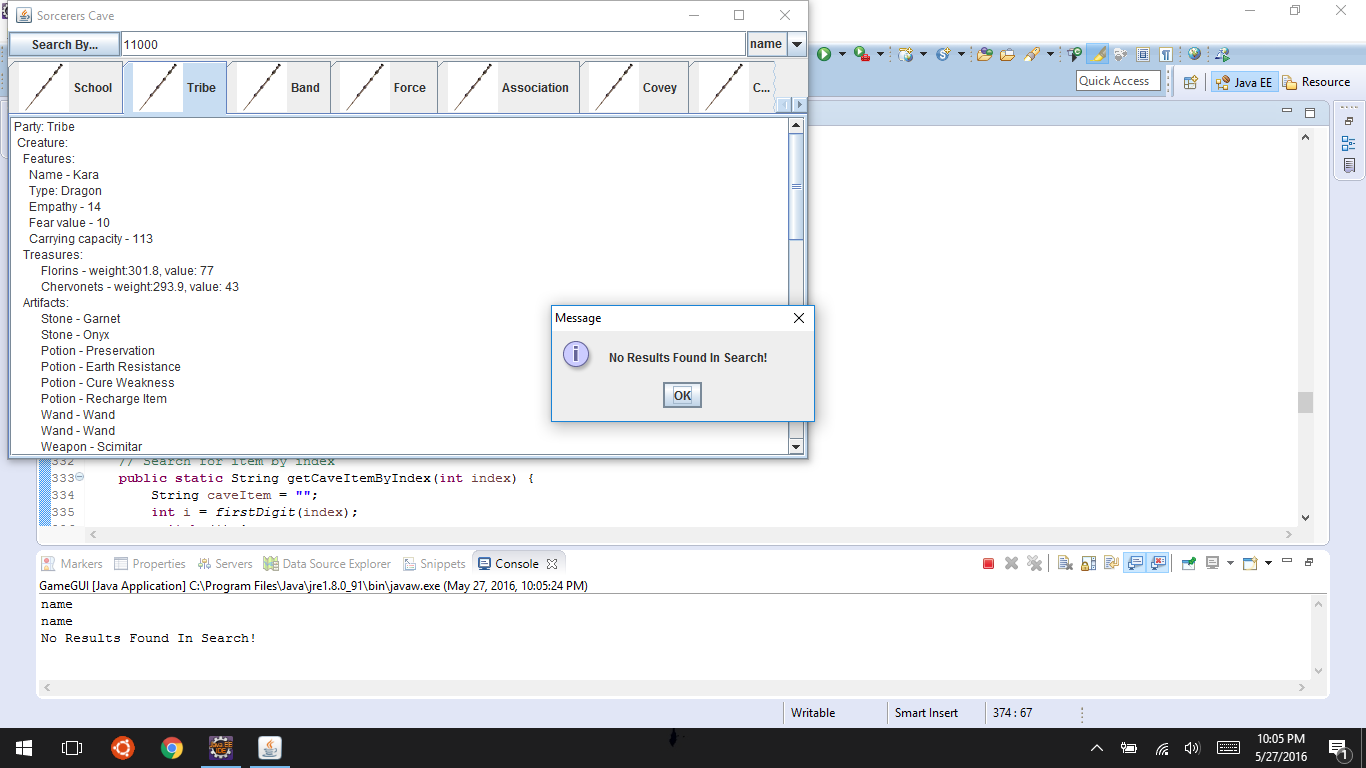
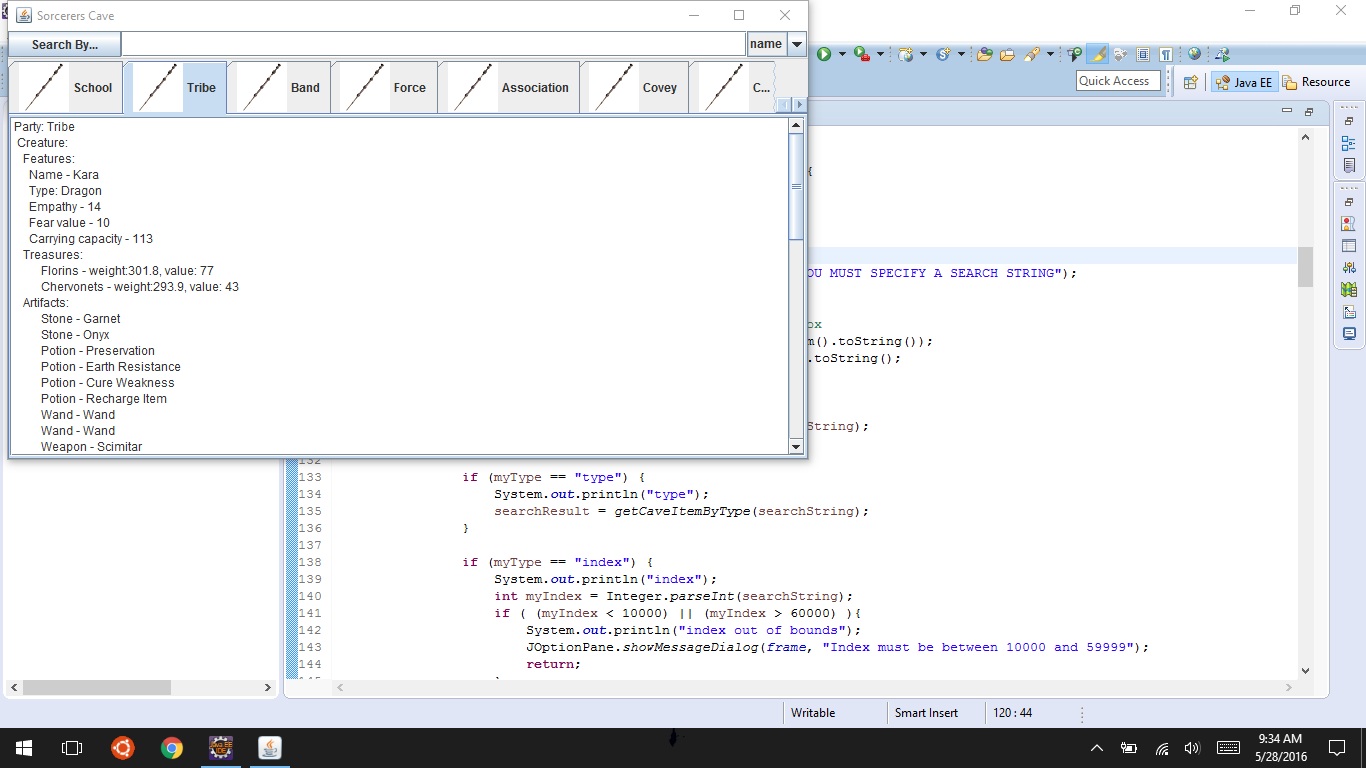


* + 1. Search by index of 20002 returns Creature Kara as expected. Pressing OK button after response given returns focus to GUI as expected.

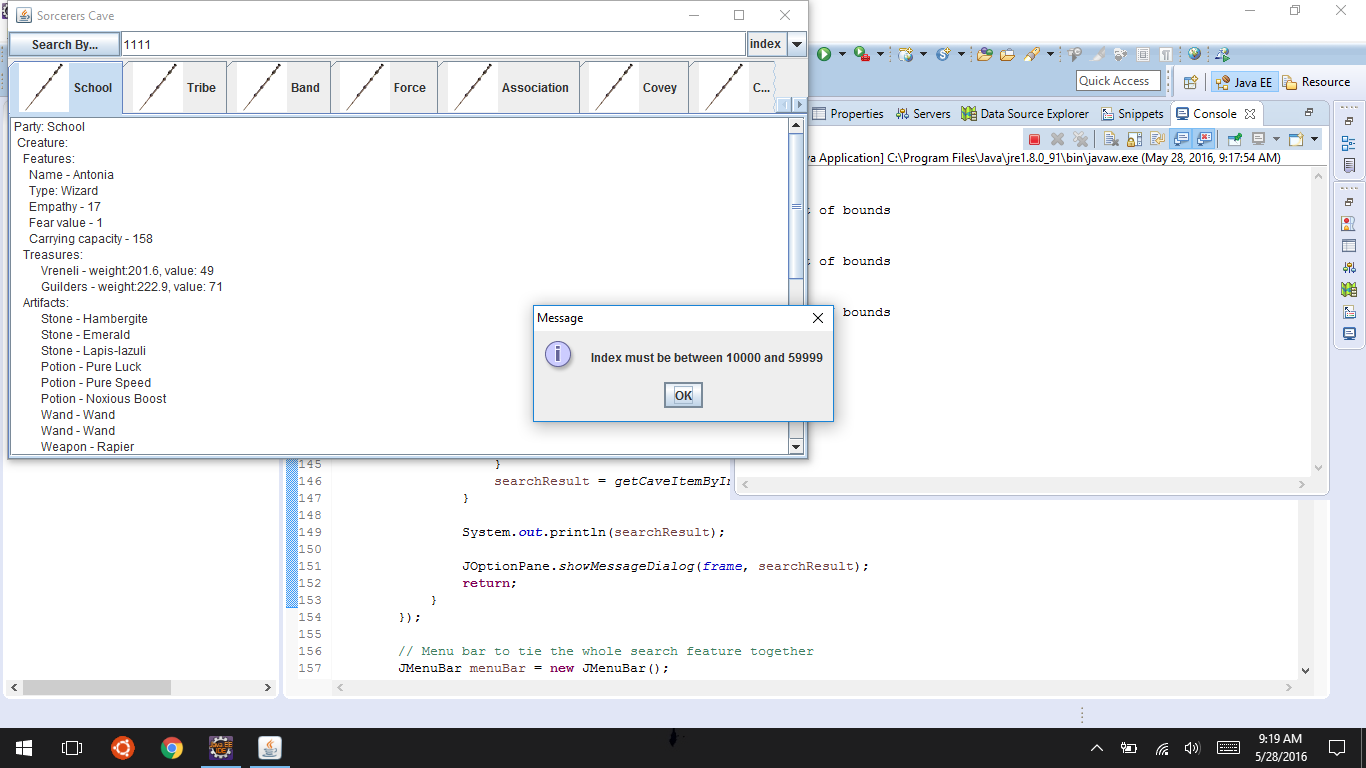


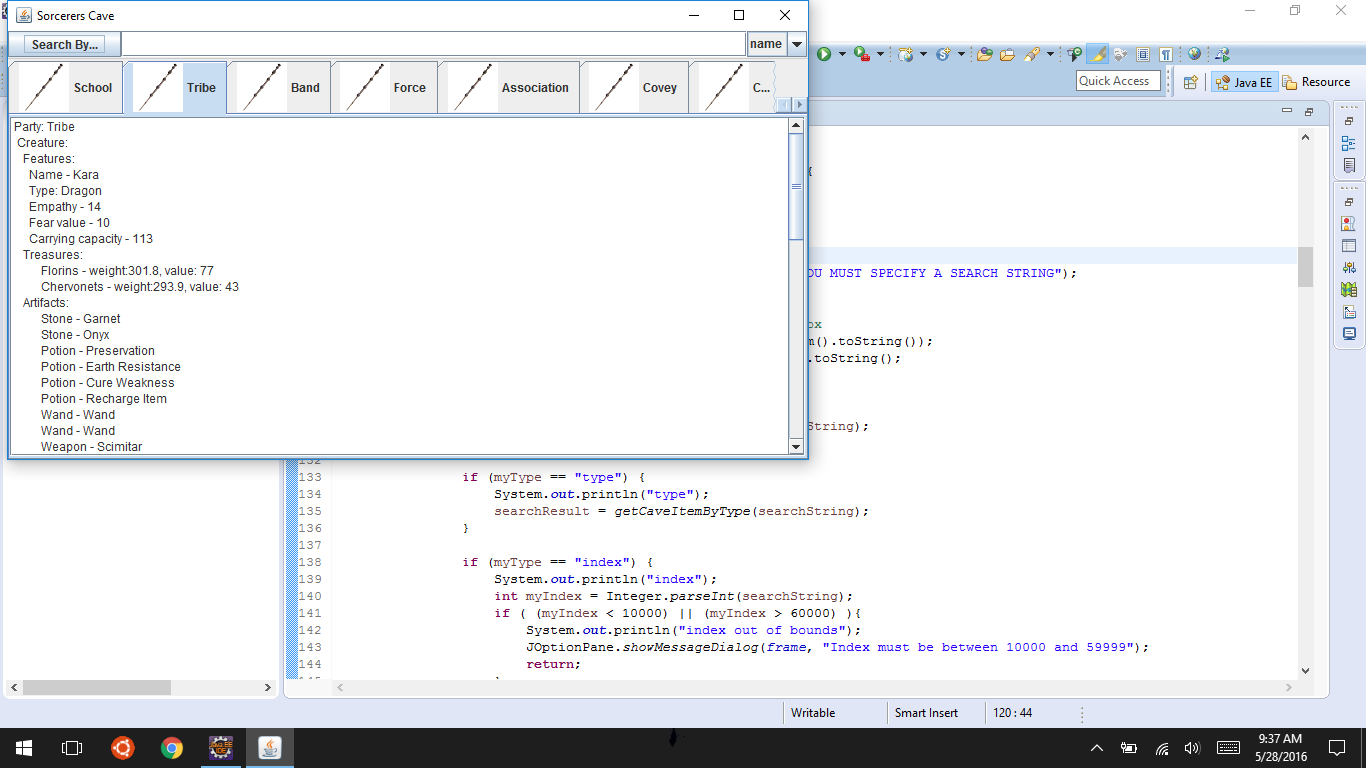


* + 1. Search of non existent index returns standard response. Pressing OK button after response given returns focus to GUI as expected.

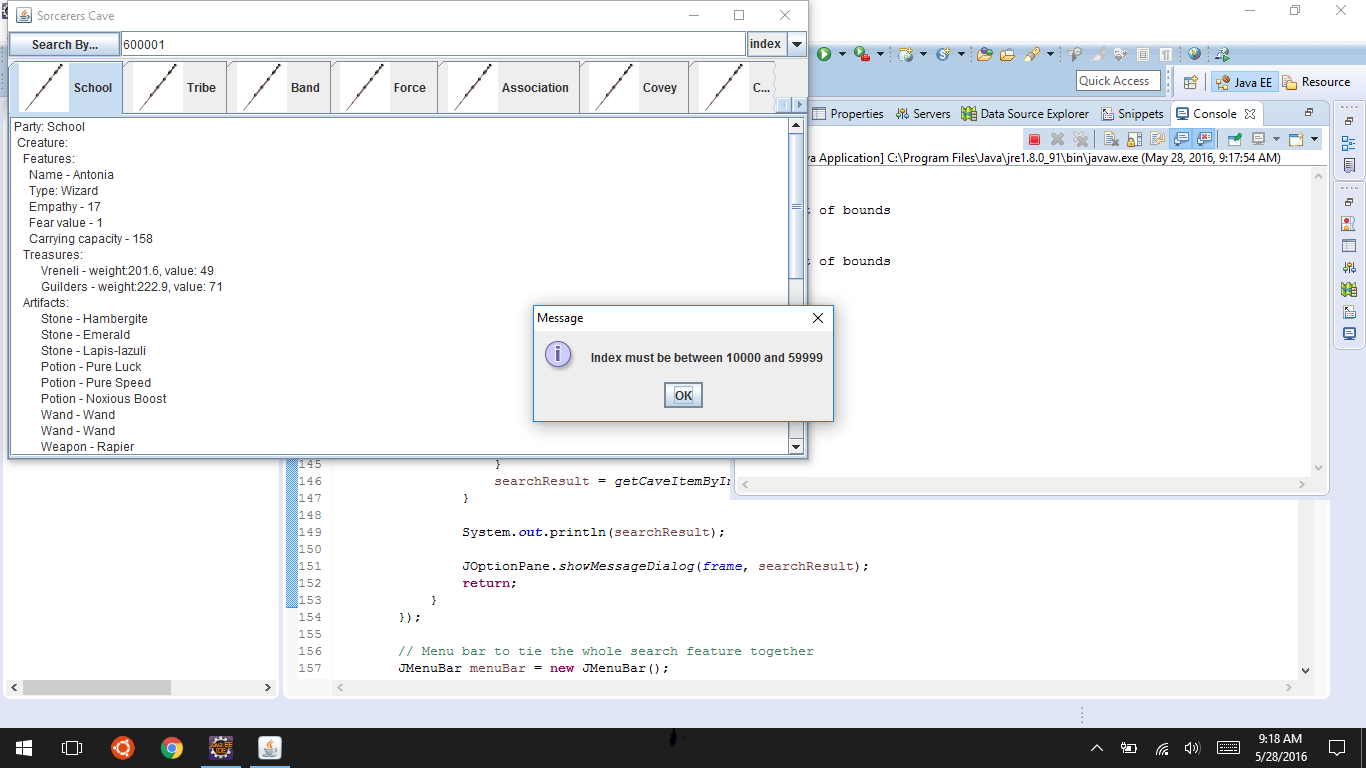
 

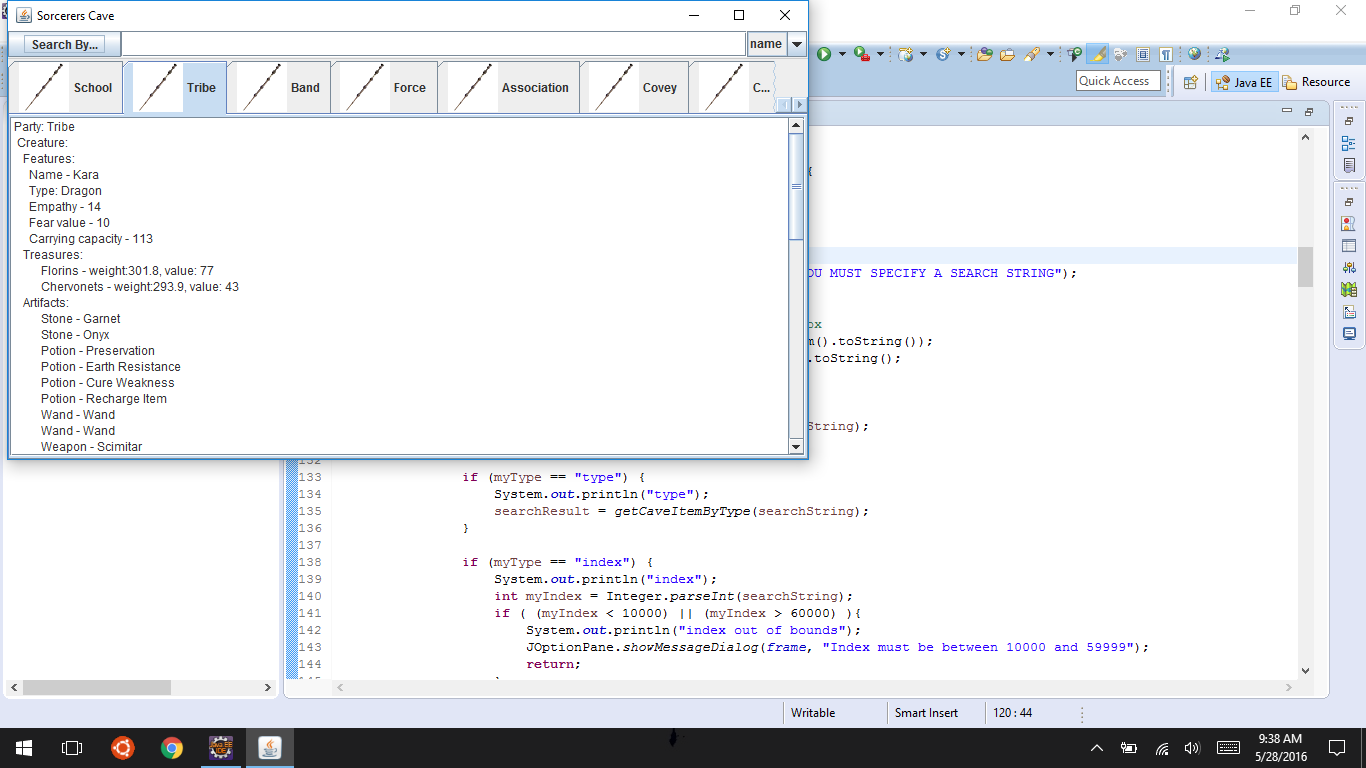
* + 1. Search of index (below lower bound) returns standardized response. Pressing OK button after response given returns focus to GUI as expected.



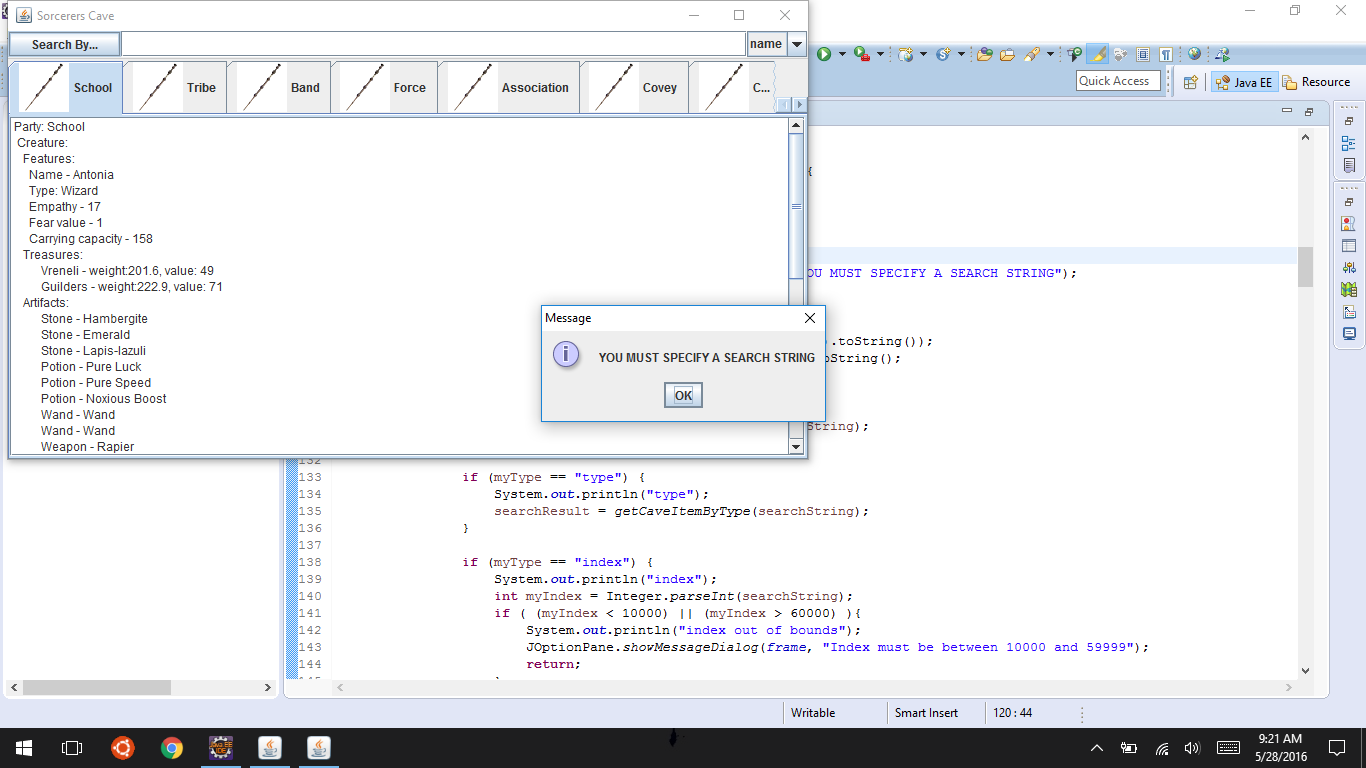


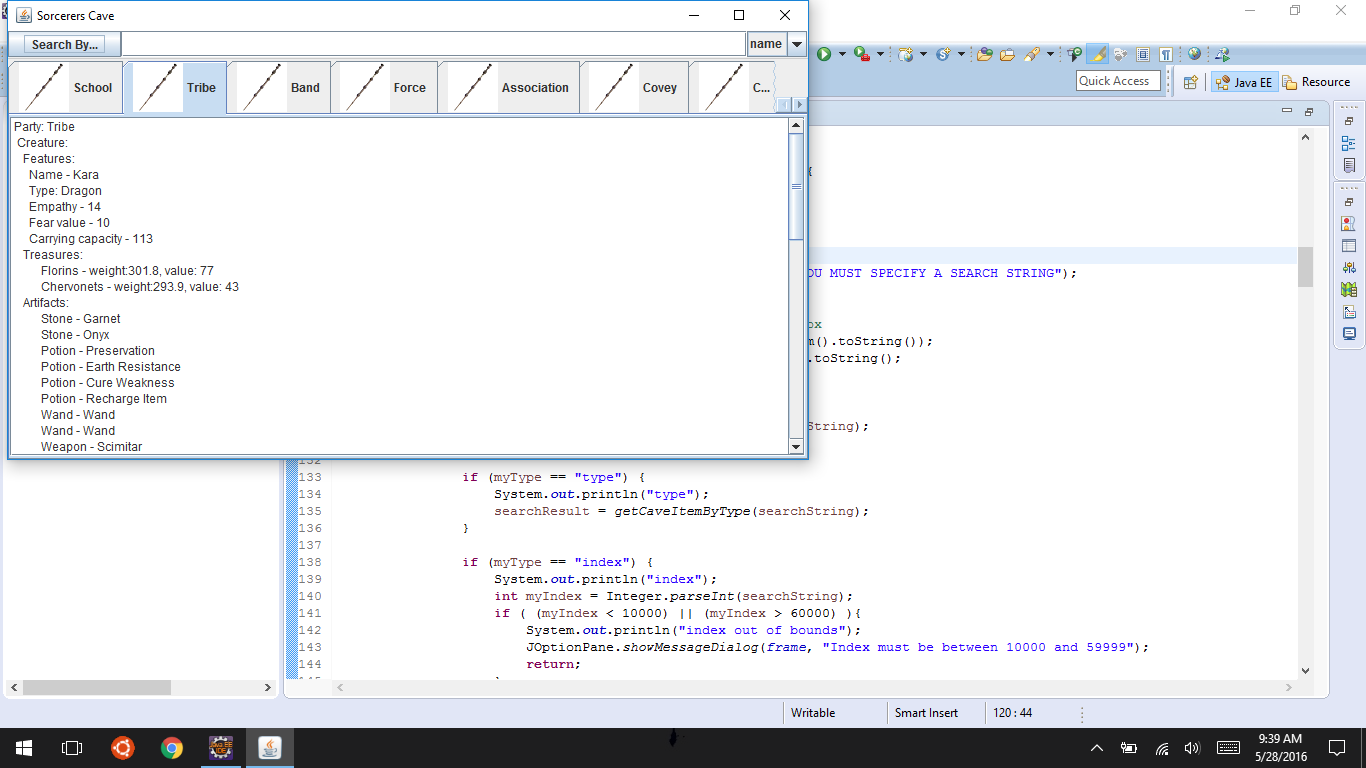
* + 1. Search of index (above upper bound) returns standardized response. Pressing OK button after response given returns focus to GUI as expected.





* + 1. Search with empty string returns correct standardized response. Pressing OK button after response given returns focus to GUI as expected.





1. Comments:
   1. I believe I would have liked to implement the Type as constants rather than strings
   2. I would be interested to try different data structures than ArrayList objects for the nested tree, as I believe other data types may be more suited for the search functionality
   3. I don’t like that I have iterative search functionality for every item, though, it is very effective for the purposes of this application since it’s not an insane amount of iteration.