SE-4831: Software Quality Assurance
Lab 7: Using Findbugs

FindBugs is a Java application which uses static analysis to look for bugs in Java code.

In this lab, you are going exercise features of the FindBugs static analysis tool.

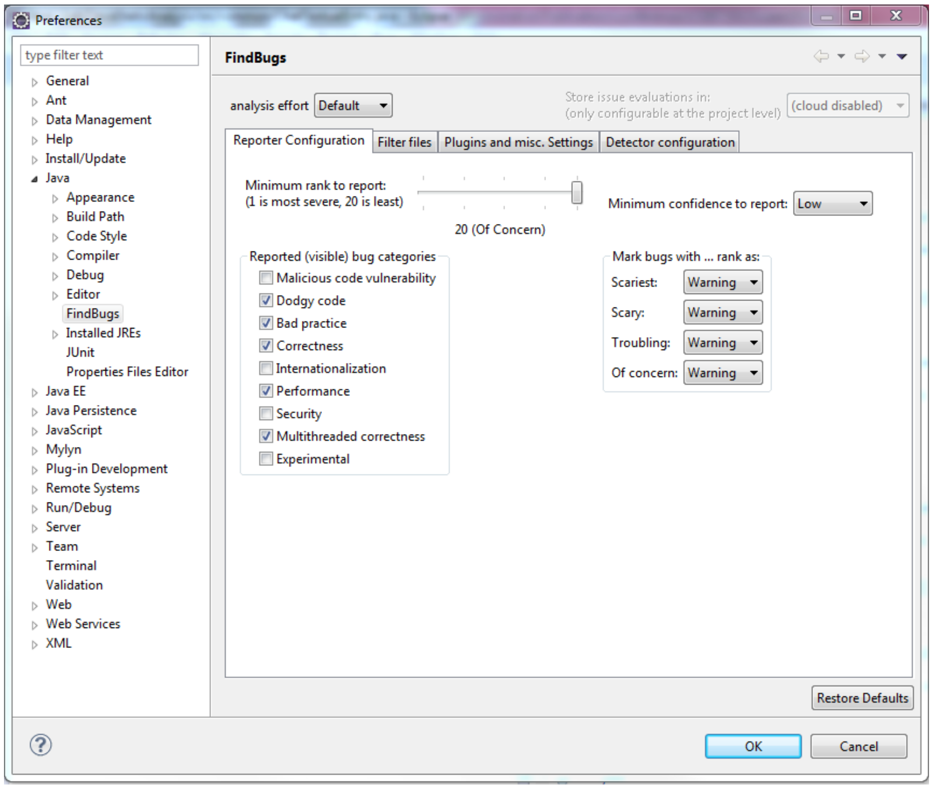
1. Introduction

As has been discussed in class, static analysis tools can be quite beneficial in the war against software failure. For this lab session, we are going to use the Findbugs tool to look at Java source code. Two pieces of code will be analyzed. One piece of code is provided by the instructor. A second piece of code is to be student provided, either from senior design projects or other course work.

2. Installing and Analyzing a known module with Findbugs

There are two approaches to installing findbugs. The first involves using Eclipse and executing Findbugs as an Eclipse plug-in. This is the preferred method. For those who do not have Eclipse, however, a separate GUI tool is available as well (or, you could run FindBugs from an Ant script.)

1. Go to the Findbugs open source site, http://findbugs.sourceforge.net. If you would like to execute the stand alone client, download the client. If you would like to run the eclipse plug in, follow the instructions for obtaining the eclipse plug in at http://findbugs.cs.umd.edu/eclipse/
2. After Eclipse is downloaded, go to the course website and download the JavaChat.zip archive. This is an Eclipse project which has source code of a known quality. (It is not perfect and does miss some coding standards issues that a true review would pick up.) Create an Eclipse project, Lab 7 (or some other name), and extract the source code into this project. A class diagram for this code is shown in the Appendix. (Note: UI classes have been omitted from the class diagram.).
3. Build the source code and make certain it cleanly builds.
4. Open the properties windows, select “Findbugs”, “Configure Workspace Settings”, and set the Minimum rank to report to 20 and “Minimum Confidence to report” to Low.



FindBugs

analysis effort: Default

Store issue evaluations in: (only configurable at the project level) (cloud disabled)

Minimum rank to report: (1 is most severe, 20 is least) 20 (Of Concern)

Minimum confidence to report: Low

Reported (visible) bug categories
- [ ] Malicious code vulnerability
- [x] Dodgy code
- [x] Bad practice
- [x] Correctness
- [ ] Internationalization
- [x] Performance
- [ ] Security
- [x] Multithreaded correctness
- [ ] Experimental

Mark bugs with ... rank as:

Scariest:	Warning
Scary:	Warning
Troubling:	Warning
Of concern:	Warning

5. Right click on the project, and select Findbugs, and analyze source code.
6. Findbugs errors will show up on the problem window. Or, if you switch to the FindBugs perspective, bugs will appear on the left.
7. Analyze each of the bugs that was found. Is it valid? Is it invalid? Overall, how many bugs were found? How many were valid? How many were invalid? Would these have been easily caught in a code review? How much longer would a code review have taken? Is this easier than if you had done this manually? Can you easily fix these problems now that they have been identified?
8. From the problems window on the bottom of the screen, select the bugs and copy them to the clipboard. Open an Excel spreadsheet and paste them into a table. This will allow you to have a nice formatted table for inclusion in reports.

3. Part 2: Analyzing your own code

Now that you have analyzed a fixed segment of source code, you are to select a piece of your own code and perform the same analysis on it. The module you select can be part of Senior design, something from another class, or something you have worked on independently. Follow essentially the same process.

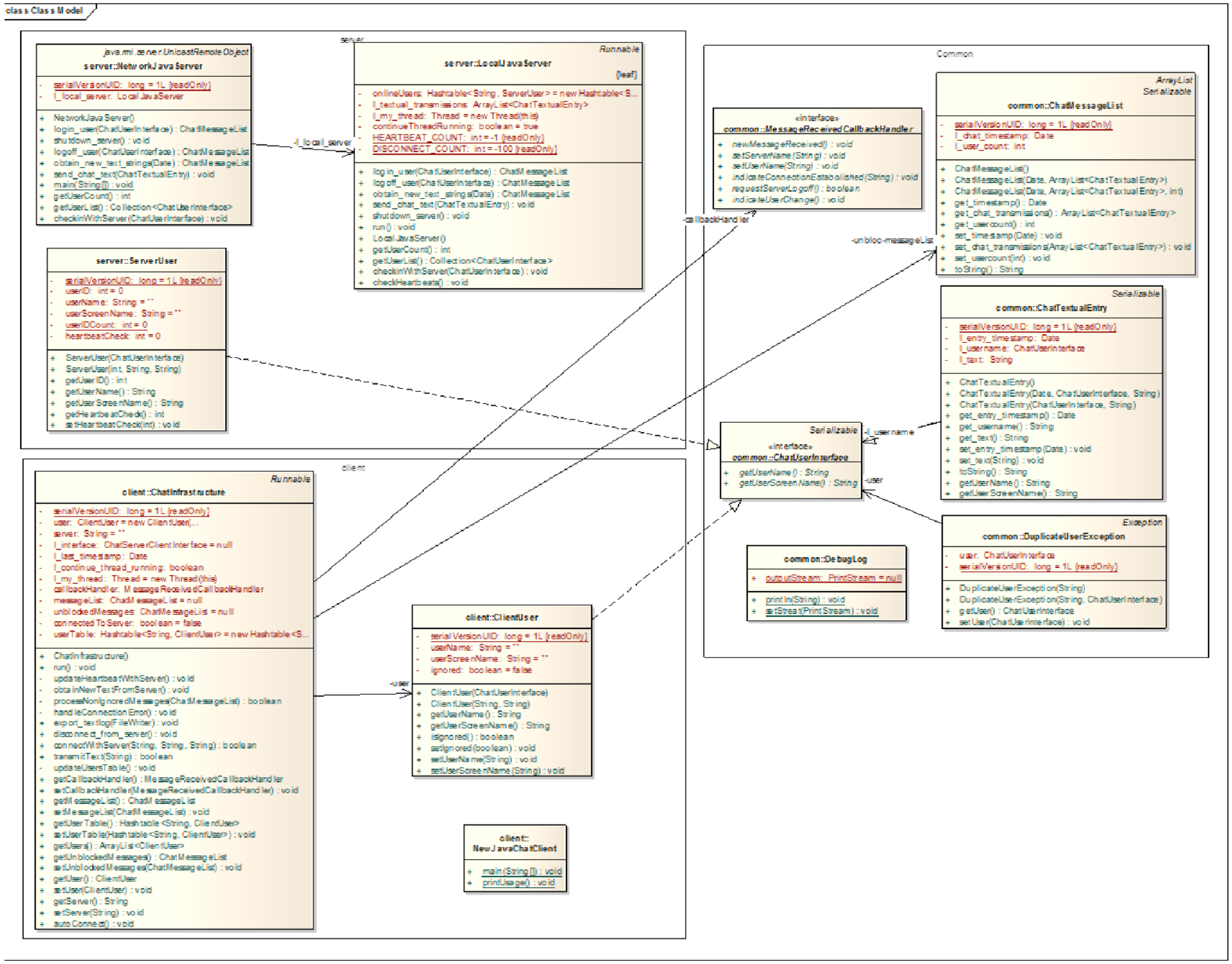
4. Lab Deliverables

By the 23:59 on February 4, 2014, you should submit the following report using the Web upload script:

1. Introduction
    a. What did you do with this lab?
2. Chat Tool Analysis -> Using the analysis of the provided chat tool, answer the following:
    a. What did you find with the Chat tool?
    b. How effective were the analysis tools? Were the majority of the findings valid or were they false positives?
    c. Did the tools find the same things you would have found had a formal inspection been done on the code?
    d. If you had done a formal inspection on this code, based on its size, how long would it have taken?[1]
    e. Include a table of the found bugs.
3. Code Analysis of your source code
    a. Describe what the source code is that you analyzed. What is it and what is it used for?
    b. How big is the code you analyzed
    c. How mature is the code you are analyzing? Has it been fully tested and deployed, or has it simply been developed?
    d. What did you find lurking in your code?
    e. How effective were the analysis tools?
    f. Did the tool find the same things you would have found had a formal inspection been done on the code?
    g. If you had done a formal inspection on this code, based on its size, how long would it have taken?[2]
    h. Include a table of the bugs that were found.
4. Things gone right / Things gone wrong
    a. This section shall discuss the things which went correctly with this lab well as the things which posed problems during this lab.
5. Conclusions -> This section shall discuss what has been learned from this laboratory experience.
    a. What do you now know about the capabilities of static analysis tools?
    b. Do these tools offer potential within your development process?
    c. What are the limitations of this class of tools that this lab has shown you?

If you have any questions, consult your instructor.

[1] Note: You can use any LOC counter of choice.
[2] Note: You can use any LOC counter of choice.


class Class Model
server
java.rmi.server.UnicastRemoteObject
server::NetworkJavaServer
- serialVersionUID: long = 1L {readOnly}
- l_local_server: LocalJavaServer
+ NetworkJavaServer()
+ login_user(ChatUserInterface) : ChatMessageList
+ shutdown_server() : void
+ logoff_user(ChatUserInterface) : ChatMessageList
+ obtain_new_text_strings(Date) : ChatMessageList
+ send_chat_text(ChatTextualEntry) : void
+ main(String[]) : void
+ getUserCount() : int
+ getUserList() : Collection<ChatUserInterface>
+ checkinWithServer(ChatUserInterface) : void
-l_local_server
Runnable
server::LocalJavaServer
{leaf}
- onlineUsers: Hashtable<String, ServerUser> = new Hashtable<S...
- l_textual_transmissions: ArrayList<ChatTextualEntry>
- l_my_thread: Thread = new Thread(this)
- continueThreadRunning: boolean = true
- HEARTBEAT_COUNT: int = -1 {readOnly}
- DISCONNECT_COUNT: int = -100 {readOnly}
+ login_user(ChatUserInterface) : ChatMessageList
+ logoff_user(ChatUserInterface) : ChatMessageList
+ obtain_new_text_strings(Date) : ChatMessageList
+ send_chat_text(ChatTextualEntry) : void
+ shutdown_server() : void
+ run() : void
+ LocalJavaServer()
+ getUserCount() : int
+ getUserList() : Collection<ChatUserInterface>
+ checkinWithServer(ChatUserInterface) : void
+ checkHeartbeats() : void
server::ServerUser
- serialVersionUID: long = 1L {readOnly}
- userID: int = 0
- userName: String = ""
- userScreenName: String = ""
- userIDCount: int = 0
- heartbeatCheck: int = 0
+ ServerUser(ChatUserInterface)
+ ServerUser(int, String, String)
+ getUserID() : int
+ getUserName() : String
+ getUserScreenName() : String
+ getHeartbeatCheck() : int
+ setHeartbeatCheck(int) : void
client
Runnable
client::ChatInfrastructure
- serialVersionUID: long = 1L {readOnly}
- user: ClientUser = new ClientUser(...
- server: String = ""
- l_interface: ChatServerClientInterface = null
- l_last_timestamp: Date
- l_continue_thread_running: boolean
- l_my_thread: Thread = new Thread(this)
- callbackHandler: MessageReceivedCallbackHandler
- messageList: ChatMessageList = null
- unblockedMessages: ChatMessageList = null
- connectedToServer: boolean = false
- userTable: Hashtable<String, ClientUser> = new Hashtable<S...
+ ChatInfrastructure()
+ run() : void
- updateHeartbeatWithServer() : void
- obtainNewTextFromServer() : void
- processNonIgnoredMessages(ChatMessageList) : boolean
- handleConnectionError() : void
+ export_textlog(FileWriter) : void
+ disconnect_from_server() : void
+ connectWithServer(String, String, String) : boolean
+ transmitText(String) : boolean
- updateUsersTable() : void
+ getCallbackHandler() : MessageReceivedCallbackHandler
+ setCallbackHandler(MessageReceivedCallbackHandler) : void
+ getMessageList() : ChatMessageList
+ setMessageList(ChatMessageList) : void
+ getUserTable() : Hashtable<String, ClientUser>
+ setUserTable(Hashtable<String, ClientUser>) : void
+ getUsers() : ArrayList<ClientUser>
+ getUnblockedMessages() : ChatMessageList
+ setUnblockedMessages(ChatMessageList) : void
+ getUser() : ClientUser
+ setUser(ClientUser) : void
+ getServer() : String
+ setServer(String) : void
+ autoConnect() : void
-user
client::ClientUser
- serialVersionUID: long = 1L {readOnly}
- userName: String = ""
- userScreenName: String = ""
- ignored: boolean = false
+ ClientUser(ChatUserInterface)
+ ClientUser(String, String)
+ getUserName() : String
+ getUserScreenName() : String
+ isIgnored() : boolean
+ setIgnored(boolean) : void
+ setUserName(String) : void
+ setUserScreenName(String) : void
client::
NewJavaChatClient
+ main(String[]) : void
+ printUsage() : void
Common
«interface»
common::MessageReceivedCallbackHandler
+ newMessageReceived() : void
+ setServerName(String) : void
+ setUserName(String) : void
+ indicateConnectionEstablished(String) : void
+ requestServerLogoff() : boolean
+ indicateUserChange() : void
-callbackHandler
-unblockedMessageList
ArrayList
Serializable
common::ChatMessageList
- serialVersionUID: long = 1L {readOnly}
- l_chat_timestamp: Date
- l_user_count: int
+ ChatMessageList()
+ ChatMessageList(Date, ArrayList<ChatTextualEntry>)
+ ChatMessageList(Date, ArrayList<ChatTextualEntry>, int)
+ get_timestamp() : Date
+ get_chat_transmissions() : ArrayList<ChatTextualEntry>
+ get_usercount() : int
+ set_timestamp(Date) : void
+ set_chat_transmissions(ArrayList<ChatTextualEntry>) : void
+ set_usercount(int) : void
+ toString() : String
Serializable
common::ChatTextualEntry
- serialVersionUID: long = 1L {readOnly}
- l_entry_timestamp: Date
- l_username: ChatUserInterface
- l_text: String
+ ChatTextualEntry()
+ ChatTextualEntry(Date, ChatUserInterface, String)
+ ChatTextualEntry(ChatUserInterface, String)
+ get_entry_timestamp() : Date
+ get_username() : String
+ get_text() : String
+ set_entry_timestamp(Date) : void
+ set_text(String) : void
+ toString() : String
+ getUserName() : String
+ getUserScreenName() : String
-l_username
Serializable
«interface»
common::ChatUserInterface
+ getUserName() : String
+ getUserScreenName() : String
-user
Exception
common::DuplicateUserException
- user: ChatUserInterface
- serialVersionUID: long = 1L {readOnly}
+ DuplicateUserException(String)
+ DuplicateUserException(String, ChatUserInterface)
+ getUser() : ChatUserInterface
+ setUser(ChatUserInterface) : void
common::DebugLog
+ outputStream: PrintStream = null
+ println(String) : void
+ setStream(PrintStream) : void