CS102	**Spring 2018/19**	Project Group	3B
Instructor:	**David Davenport**		
Assistant:	**Mehmet Başaran**		

~ Nito ~

Criteria	TA/Grader	Instructor
Presentation		
Overall		

Internationals

Ziya Mukhtarov, Mokhlaroyim Raupova, Javid Baghirov, Mannan Abdul, Alper Sari, Adeem Adil Khatri

Project UI Design Report

(Version 2.0)

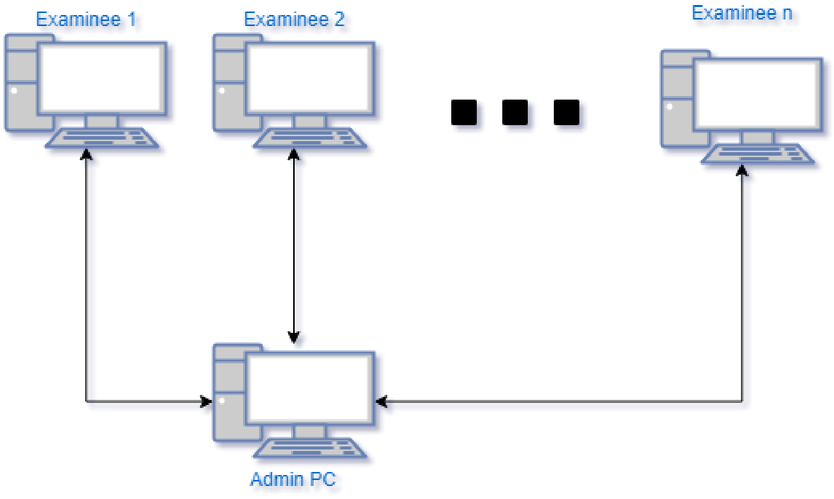
14 May 2019

1. Introduction

Nito – the exam system aims to provide its users with a modern and elegant solution to digitalize the old-fashioned tools (e.g. paper, pencil) used during programming exams. Using al-most all features that technology and the internet provide us with, Nito strives to fill this gap as unfortunately there is almost no application which gathers such anti-cheating services into one software.

2. System Overview

2.1 Network



This is the basic network diagram showing the connections between Nito interfaces during an exam. The first version of the program will only support one administrator for each exam. That admin interface will be connected to all examinee interfaces via both UDP and TCP protocols. UDP protocol will be used for live screen view feature, while all other requests will be done using TCP as it is more reliable than UDP.
The first version will only be available for the Windows operating system. The admin PC will function as a server, while examinees will be connected as clients.
All relevant classes will probably implement Java’s Serializable interface in order to be sent over the network.

2.2 Storage

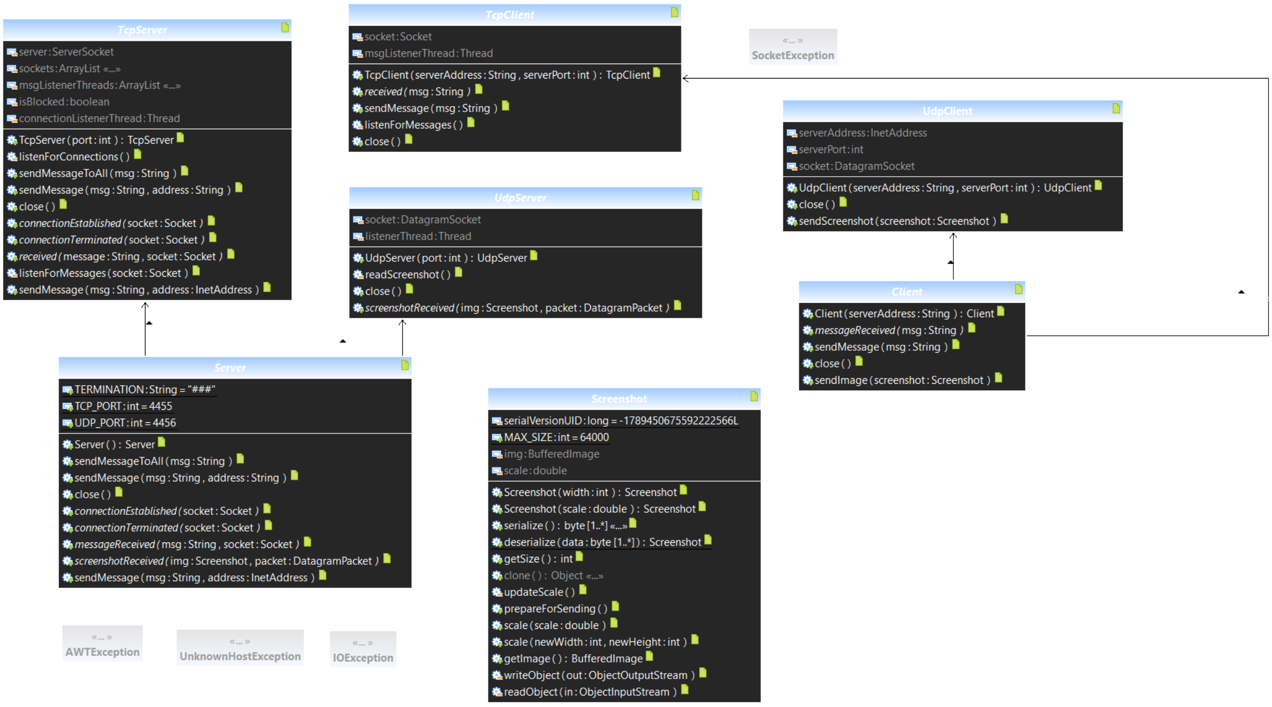
Nito will use file-based storage. All of the exam data will be stored in the Admin PC. The relevant data will flow from admin to examinees and kept in a temporary folder until the exam ends. The progress of examinees will be backed up to their memory continuously. Once the exam ends or submit action occurs, all of the solutions will be transferred to admin pc and stored in folders.

2.3 Graphical User Interface (GUI)

For designing GUIs, JavaFX library was used. As a part of the process, FXML and CSS languages were also used and had to be learned by the group members who were associated with designing the GUI . Nito interface will use third-party layout managers for some complex layouts such as MigLayout. For some features, Swing components might be integrated into JavaFX as it lacks some rare components (e.g. JDesktopPane).

3. Core Design Details

3.1 network Package



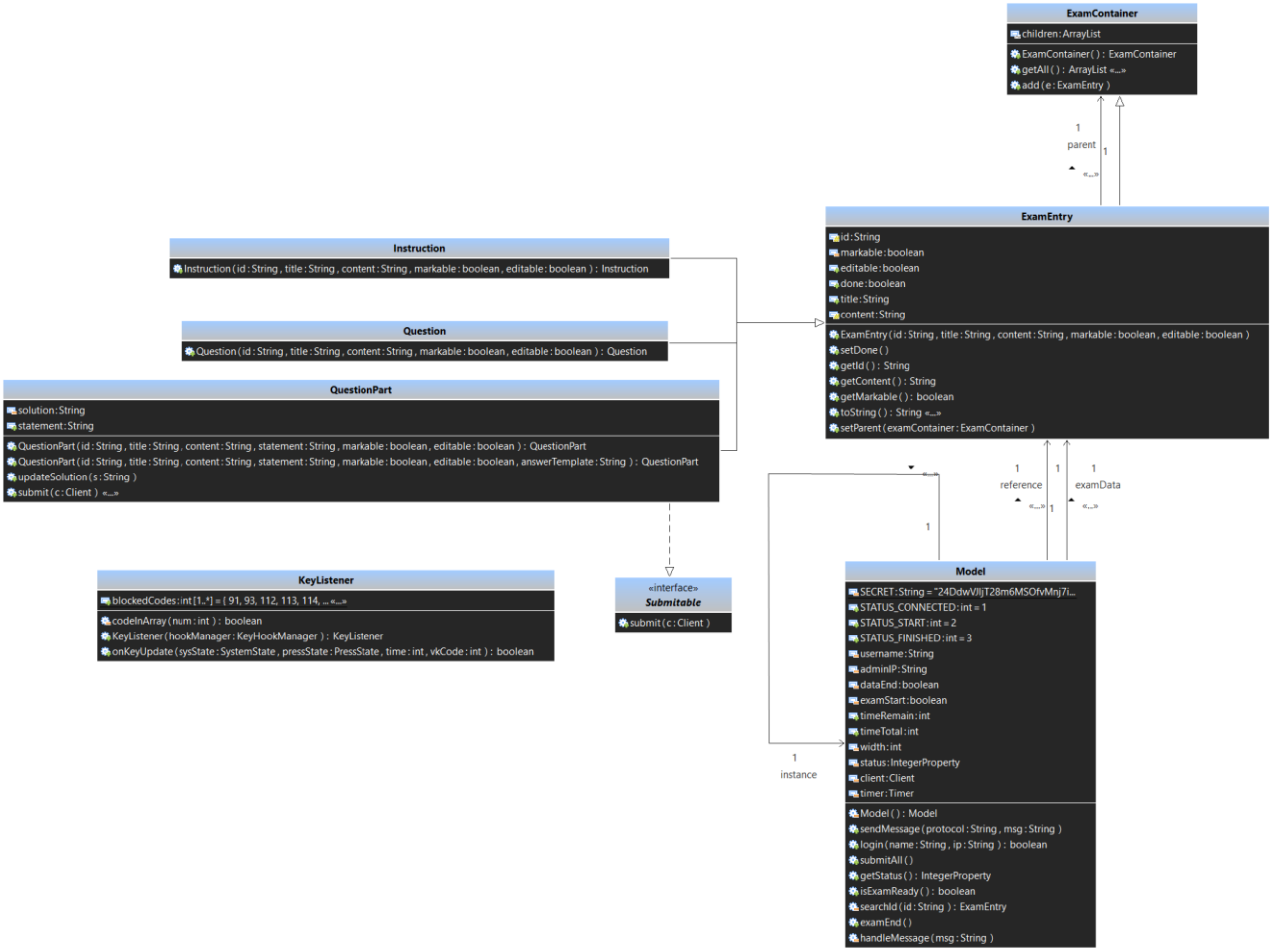
TcpServer
server:ServerSocket
sockets:ArrayList «...»
msgListenerThreads:ArrayList «...»
isBlocked:boolean
connectionListenerThread:Thread
TcpServer (port : int) : TcpServer
listenForConnections ( )
sendMessageToAll (msg : String)
sendMessage (msg : String , address : String)
close ( )
connectionEstablished (socket : Socket)
connectionTerminated (socket : Socket)
received (message : String , socket : Socket)
listenForMessages (socket : Socket)
sendMessage (msg : String , address : InetAddress)
TcpClient
socket:Socket
msgListenerThread:Thread
TcpClient (serverAddress : String , serverPort : int) : TcpClient
received (msg : String)
sendMessage (msg : String)
listenForMessages ( )
close ( )
«...»
SocketException
UdpClient
serverAddress:InetAddress
serverPort:int
socket:DatagramSocket
UdpClient (serverAddress : String , serverPort : int) : UdpClient
close ( )
sendScreenshot (screenshot : Screenshot)
UdpServer
socket:DatagramSocket
listenerThread:Thread
UdpServer (port : int) : UdpServer
readScreenshot ( )
close ( )
screenshotReceived (img : Screenshot , packet : DatagramPacket)
Client
Client (serverAddress : String) : Client
messageReceived (msg : String)
sendMessage (msg : String)
close ( )
sendImage (screenshot : Screenshot)
Server
TERMINATION:String = "###"
TCP_PORT:int = 4455
UDP_PORT:int = 4456
Server ( ) : Server
sendMessageToAll (msg : String)
sendMessage (msg : String , address : String)
close ( )
connectionEstablished (socket : Socket)
connectionTerminated (socket : Socket)
messageReceived (msg : String , socket : Socket)
screenshotReceived (img : Screenshot , packet : DatagramPacket)
sendMessage (msg : String , address : InetAddress)
Screenshot
serialVersionUID:long = -1789450675592222566L
MAX_SIZE:int = 64000
img:BufferedImage
scale:double
Screenshot (width : int) : Screenshot
Screenshot (scale : double) : Screenshot
serialize ( ) : byte [1..*] «...»
deserialize (data : byte [1..*]) : Screenshot
getSize ( ) : int
clone ( ) : Object «...»
updateScale ( )
prepareForSending ( )
scale (scale : double)
scale (newWidth : int , newHeight : int)
getImage ( ) : BufferedImage
writeObject (out : ObjectOutputStream)
readObject (in : ObjectInputStream)
«...»
AWTException
«...»
UnknownHostException
«...»
IOException

Both Server and Client classes contain TCP and UDP versions. Admin interface will use the Server class, while examinee interfaces will use the Client class to communicate over the

Network. Screenshot class is a wrapper class for BufferedImage and other relevant data. It also provides functionalities needed for Nito application

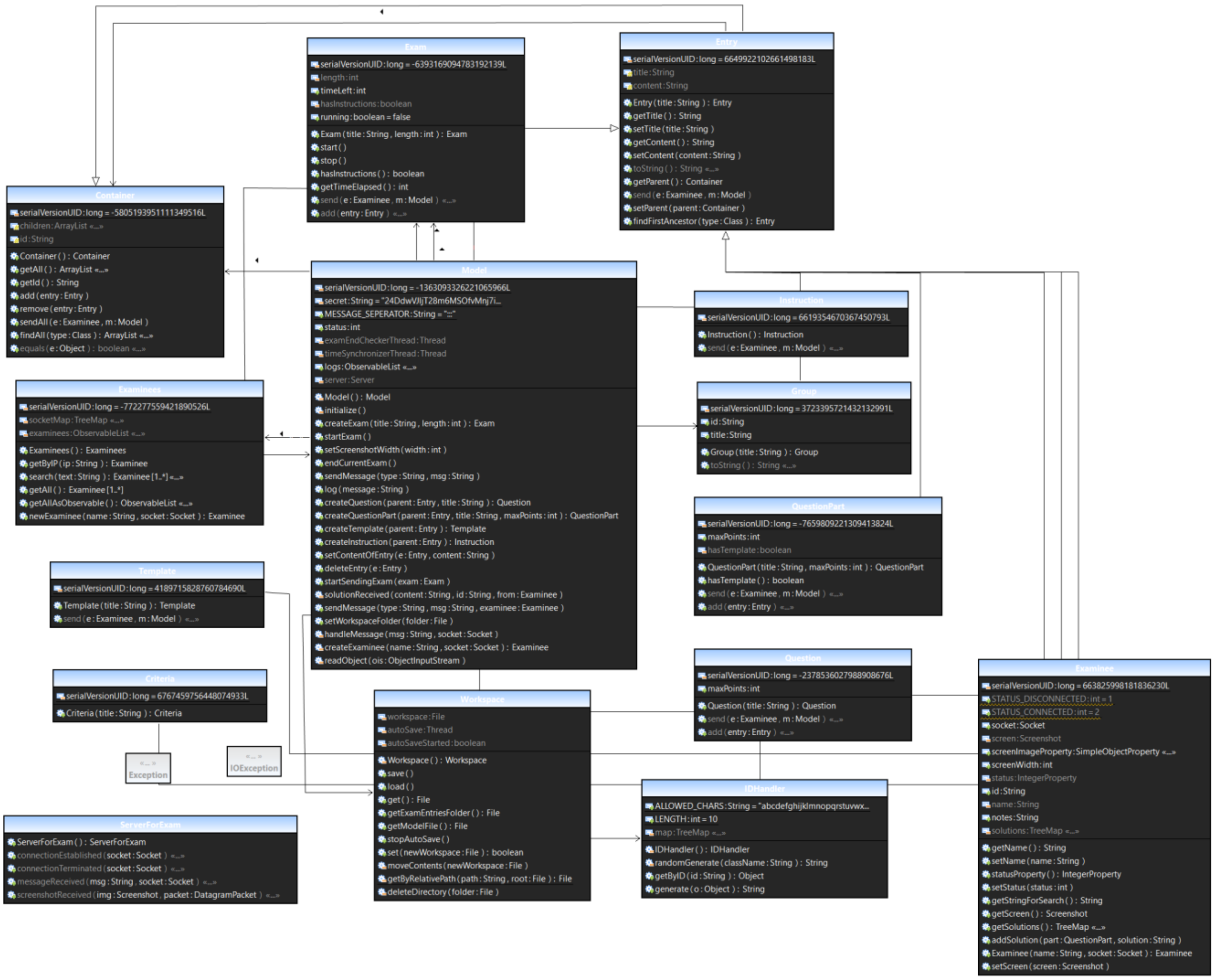
The network package is also responsible for handling all the communications between the examinees and the administrator they are connected to. This includes getting and transmitting the screenshots of the examinee screens so that the admin can monitor the exams. All exam data including the exam questions and the related info along with the solutions of the examinees and any message that the admin wants to send to an examinee, all of these will use the network model above to complete their functions.

3.2 `examinee` Package


ExamContainer
children:ArrayList
ExamContainer ( ) : ExamContainer
getAll ( ) : ArrayList «...»
add (e : ExamEntry)
1
parent
1
«...»
ExamEntry
id:String
markable:boolean
editable:boolean
done:boolean
title:String
content:String
ExamEntry (id : String , title : String , content : String , markable : boolean , editable : boolean)
setDone ( )
getId ( ) : String
getContent ( ) : String
getMarkable ( ) : boolean
toString ( ) : String «...»
setParent (examContainer : ExamContainer)
Instruction
Instruction (id : String , title : String , content : String , markable : boolean , editable : boolean) : Instruction
Question
Question (id : String , title : String , content : String , markable : boolean , editable : boolean) : Question
QuestionPart
solution:String
statement:String
QuestionPart (id : String , title : String , content : String , statement : String , markable : boolean , editable : boolean) : QuestionPart
QuestionPart (id : String , title : String , content : String , statement : String , markable : boolean , editable : boolean , answerTemplate : String) : QuestionPart
updateSolution (s : String)
submit (c : Client) «...»
«...»
1
1
reference
«...»
1
1
1
examData
«...»
KeyListener
blockedCodes:int [1..*] = { 91, 93, 112, 113, 114, ... «...»
codeInArray (num : int) : boolean
KeyListener (hookManager : KeyHookManager) : KeyListener
onKeyUpdate (sysState : SystemState , pressState : PressState , time : int , vkCode : int) : boolean
«interface»
Submitable
submit (c : Client)
Model
SECRET:String = "24DdwVJIjT28m6MSOfvMnj7i...
STATUS_CONNECTED:int = 1
STATUS_START:int = 2
STATUS_FINISHED:int = 3
username:String
adminIP:String
dataEnd:boolean
examStart:boolean
timeRemain:int
timeTotal:int
width:int
status:IntegerProperty
client:Client
timer:Timer
Model ( ) : Model
sendMessage (protocol : String , msg : String)
login (name : String , ip : String) : boolean
submitAll ( )
getStatus ( ) : IntegerProperty
isExamReady ( ) : boolean
searchId (id : String) : ExamEntry
examEnd ( )
handleMessage (msg : String)
1
instance

These are the classes for the examinee interface. ExamEntry and ExamContainer classes are similar to Swing classes in the sense that it is possible to create complex entries because each entry is a container as is the case with the admin model following a Swing like logic, Question and Question Parts extend ExamEntry to be to the tree view of exams. Also, only QuestionParts class implements Submitable as only question parts can be submitted during the exam. The class KeyListener is a separate class as it blocks the all shortcuts using CTRL, ALT, WIN, ESC, and function keys. On the other hand, to avoid multiple logins from a single device, Model class was created as a singleton class.

3.3 admin package


Exam
serialVersionUID:long = -6393169094783192139L
length:int
timeLeft:int
hasInstructions:boolean
running:boolean = false
Exam (title : String , length : int) : Exam
start ( )
stop ( )
hasInstructions ( ) : boolean
getTimeElapsed ( ) : int
send (e : Examinee , m : Model) «...»
add (entry : Entry) «...»
Entry
serialVersionUID:long = 6649922102661498183L
title:String
content:String
Entry (title : String) : Entry
getTitle ( ) : String
setTitle (title : String)
getContent ( ) : String
setContent (content : String)
toString ( ) : String «...»
getParent ( ) : Container
send (e : Examinee , m : Model)
setParent (parent : Container)
findFirstAncestor (type : Class) : Entry
Container
serialVersionUID:long = -5805193951111349516L
children:ArrayList «...»
id:String
Container ( ) : Container
getAll ( ) : ArrayList «...»
getId ( ) : String
add (entry : Entry)
remove (entry : Entry)
sendAll (e : Examinee , m : Model)
findAll (type : Class) : ArrayList «...»
equals (e : Object) : boolean «...»
Model
serialVersionUID:long = -1363093326221065966L
secret:String = "24DdwVJljT28m6MSOfvMnj7i...
MESSAGE_SEPERATOR:String = ":::"
status:int
examEndCheckerThread:Thread
timeSynchronizerThread:Thread
logs:ObservableList «...»
server:Server
Model ( ) : Model
initialize ( )
createExam (title : String , length : int) : Exam
startExam ( )
setScreenshotWidth (width : int)
endCurrentExam ( )
sendMessage (type : String , msg : String)
log (message : String)
createQuestion (parent : Entry , title : String) : Question
createQuestionPart (parent : Entry , title : String , maxPoints : int) : QuestionPart
createTemplate (parent : Entry) : Template
createInstruction (parent : Entry) : Instruction
setContentOfEntry (e : Entry , content : String)
deleteEntry (e : Entry)
startSendingExam (exam : Exam)
solutionReceived (content : String , id : String , from : Examinee)
sendMessage (type : String , msg : String , examinee : Examinee)
setWorkspaceFolder (folder : File)
handleMessage (msg : String , socket : Socket)
createExaminee (name : String , socket : Socket) : Examinee
readObject (ois : ObjectInputStream)
Instruction
serialVersionUID:long = 6619354670367450793L
Instruction ( ) : Instruction
send (e : Examinee , m : Model) «...»
Group
serialVersionUID:long = 3723395721432132991L
id:String
title:String
Group (title : String) : Group
toString ( ) : String «...»
Examinees
serialVersionUID:long = -7722775594218905 26L
socketMap:TreeMap «...»
examinees:ObservableList «...»
Examinees ( ) : Examinees
getByIP (ip : String) : Examinee
search (text : String) : Examinee [1..*] «...»
getAll ( ) : Examinee [1..*]
getAllAsObservable ( ) : ObservableList «...»
newExaminee (name : String , socket : Socket) : Examinee
QuestionPart
serialVersionUID:long = -7659809221309413824L
maxPoints:int
hasTemplate:boolean
QuestionPart (title : String , maxPoints : int) : QuestionPart
hasTemplate ( ) : boolean
send (e : Examinee , m : Model) «...»
add (entry : Entry) «...»
Template
serialVersionUID:long = 4189715828760784690L
Template (title : String) : Template
send (e : Examinee , m : Model) «...»
Criteria
serialVersionUID:long = 6767459756448074933L
Criteria (title : String) : Criteria
Question
serialVersionUID:long = -2378536027988908676L
maxPoints:int
Question (title : String) : Question
send (e : Examinee , m : Model) «...»
add (entry : Entry) «...»
Examinee
serialVersionUID:long = 6638259981818362 30L
STATUS_DISCONNECTED:int = 1
STATUS_CONNECTED:int = 2
socket:Socket
screen:Screenshot
screenImageProperty:SimpleObjectProperty «...»
screenWidth:int
status:IntegerProperty
id:String
name:String
notes:String
solutions:TreeMap «...»
getName ( ) : String
setName (name : String)
statusProperty ( ) : IntegerProperty
setStatus (status : int)
getStringForSearch ( ) : String
getScreen ( ) : Screenshot
getSolutions ( ) : TreeMap «...»
addSolution (part : QuestionPart , solution : String)
Examinee (name : String , socket : Socket) : Examinee
setScreen (screen : Screenshot)
«...»
Exception
«...»
IOException
Workspace
workspace:File
autoSave:Thread
autoSaveStarted:boolean
Workspace ( ) : Workspace
save ( )
load ( )
get ( ) : File
getExamEntriesFolder ( ) : File
getModelFile ( ) : File
stopAutoSave ( )
set (newWorkspace : File) : boolean
moveContents (newWorkspace : File)
getByRelativePath (path : String , root : File) : File
deleteDirectory (folder : File)
IDHandler
ALLOWED_CHARS:String = "abcdefghijklmnopqrstuvwx...
LENGTH:int = 10
map:TreeMap «...»
IDHandler ( ) : IDHandler
randomGenerate (className : String) : String
getByID (id : String) : Object
generate (o : Object) : String
ServerForExam
ServerForExam ( ) : ServerForExam
connectionEstablished (socket : Socket) «...»
connectionTerminated (socket : Socket) «...»
messageReceived (msg : String , socket : Socket) «...»
screenshotReceived (img : Screenshot , packet : DatagramPacket) «...»

These are the classes for the Admin interface. The Container and Entry classes are similar to Swing classes in the sense that it is possible to create complex entries because each entry extends a container. The exam, question, template, criteria, instruction and question part classes extend the Entry class which in turn extends the Container class as mentioned before. They follow the same Swing logic as Examinee classes. The Model class as shown above is the class that then handles all the changes that are supposed to be made to these entries including methods for creating new entries or deleting old ones among other things that are essential to the project.

3.4 External APIs

CSS – for help in designing the views and adding more customizability to the project's GUI

ControlsFX – JavaFX library that contains a lot of additional components for modern GUIs. We will use it for some interesting components that might be included in the project.

MigLayout – A very flexible and useful Layout Manager that helped extensively with it large array of customizable properties.

JNativeHook – This library provides global keyboard and mouse listeners. We used it for blocking keyboard shortcuts in examinee interface for anti-cheating

4. Task Assignments

Below are the assigned tasks for each person in our group. After a long discussion, we decided to implement only the core Nito features as we have very little time. Ziya Mukhtarov plans to improve this project during Summer and make it useable for CS101/102 courses.

Javid Baghirov	Examinee interface – GUI
Mannan Abdul	Admin interface – Exam preparation GUI
Mokhlaroyim Raupova	Admin interface – Exam Monitoring and Grading GUI
Alper Sari	Examinee interface – model classes
Adeem Adil Khatri	Admin interface – Entry and Container classes and subclasses
Ziya Mukhtarov	Admin interface – other model parts Examinee interface – anti-cheating services Network package Everything else that is not mentioned in this table and is required to run Nito

5. Conclusion

This revised report outlines how we have implemented our Project.