Name: Tyler Hofmeister ID: 10019543

CPSC 235: Final Design Document

This document outlines how my dynamic web site for CPSC 235 was made, and what the finished product looks like. The purpose of this web site is to teach the user a specific concept in the python programming language. The topic for my website is errors and exception handling.

This document goes over:

1. The overall structure of the web site.
2. The inheritance relations among classes
3. The dependencies between classes
4. The classes, methods, and their functionality
5. Comments on the creation of each of the 3 parts of the website

This document should provide a thorough explanation of the finished website, and how it works. The majority of the content is in the form of figures with explanations motivating each one.

Structure of the Website

The structure of the website has been split into three major components:

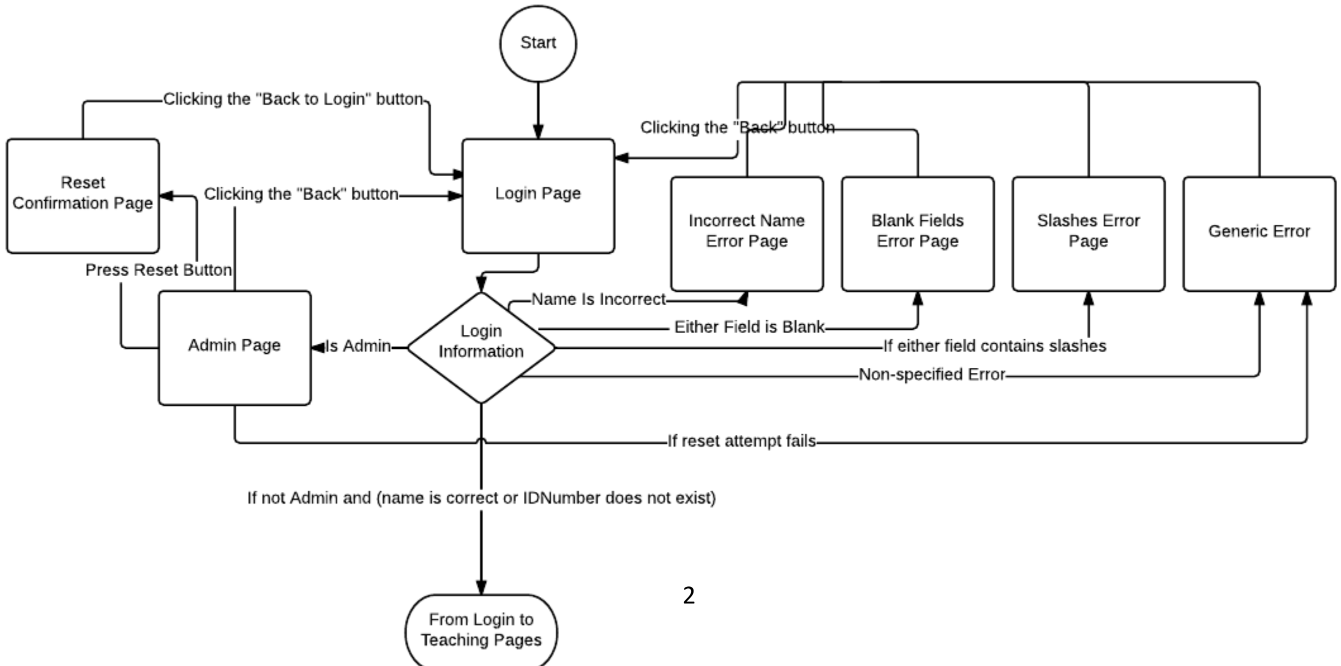
- The Login Pages
- The Teaching Pages
- The Testing Pages

Each component is designed with functionality in mind…

The Login Pages

The website login page requires users to submit both an IDNumber and a “Name” in order to use the website. These fields are used to identify the user among the others that have used the website. The backend of the website processes the submitted information. If the user filled in the fields incorrectly, or used the wrong name for an existing IDNumber, the user is directed towards an error page that informs them of their mistake. If the submitted IDNumber does not exist, a new User object is created and the user is directed towards the Teaching Pages. If the submitted IDNumber does exist and the submitted Name is correct, the user is directed towards whatever page they were on last.

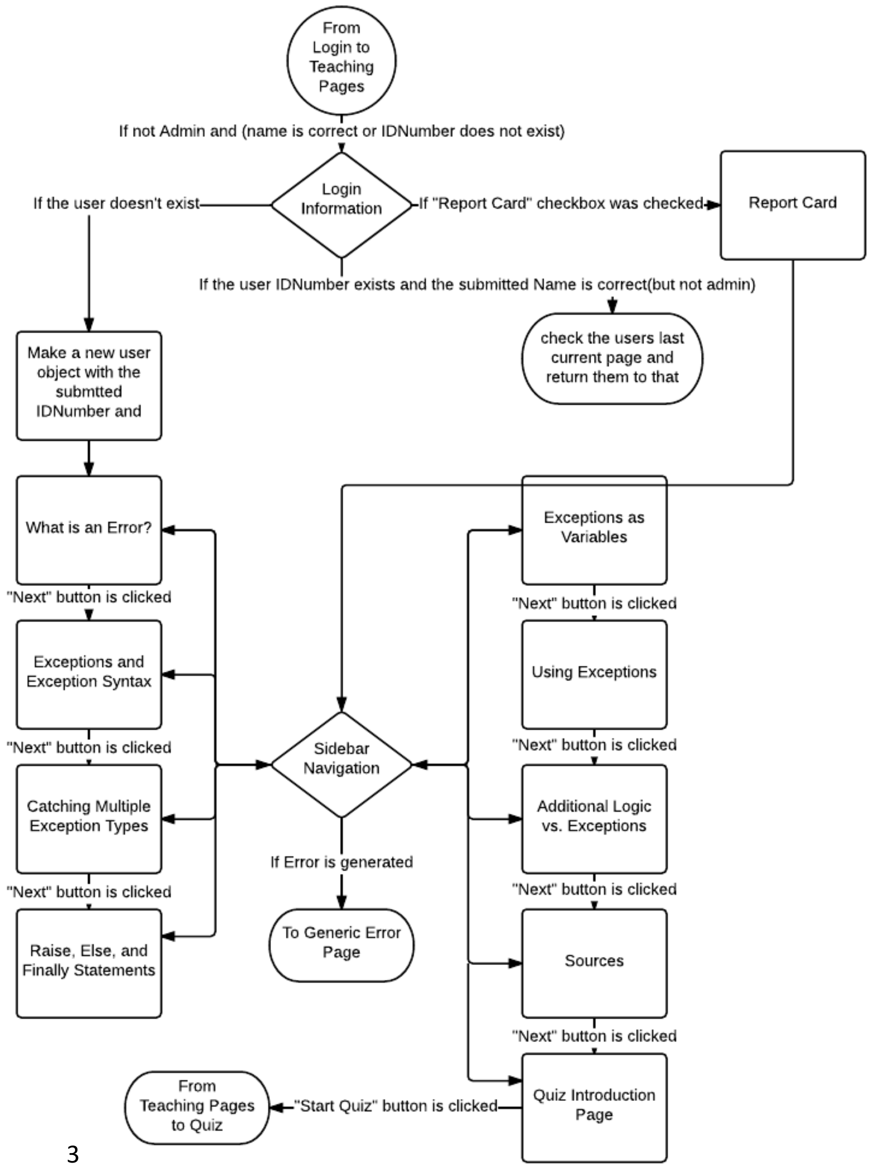
In designing the structure of the Login Pages, I focused on simplicity. Using static pages for each the different error pages ensured that the user would be provided with the information that they needed. I believed a reset confirmation page to be essential as having a user believe they had reset the website when they hadn’t would cause them a great deal of frustration. I tried to use simple “back” buttons on each page in this section so that the user would always be able to go back to the login page.


Start
Clicking the "Back to Login" button
Clicking the "Back" button
Reset Confirmation Page
Clicking the "Back" button
Login Page
Incorrect Name Error Page
Blank Fields Error Page
Slashes Error Page
Generic Error
Press Reset Button
Admin Page
Is Admin
Login Information
Name Is Incorrect
Either Field is Blank
If either field contains slashes
Non-specified Error
If reset attempt fails
If not Admin and (name is correct or IDNumber does not exist)
From Login to Teaching Pages

The Teaching Pages

The teaching pages use a sidebar style navigation so that the user can browse the pages however they want. There are also “next” buttons on the bottom of each page so that the user can progress through this part of the website in a linear manner if they want to. Note that existing users are automatically redirected back to the page they left from when they log back in. This added specification was something that I originally wanted to include. The report card is shown directly from login only if the user checks the “report card requested” checkbox on the login page. This page tracks the users progress—displaying which questions the user has answered correctly and which ones haven’t been answered correctly.

In designing the teaching pages, I wanted to be as thorough as possible. As a result, there are quite a few pages with information about what exceptions are in python and how to use them. I also wanted animations on each page that show the output of various try statements in python. The animations are simple, but when learning programming I found that seeing the output of the code was a very helpful tool to learning what various statements did. With my provided examples, I wanted to use a single idea for each example and have all of the examples build off this one train of thought as much as possible.

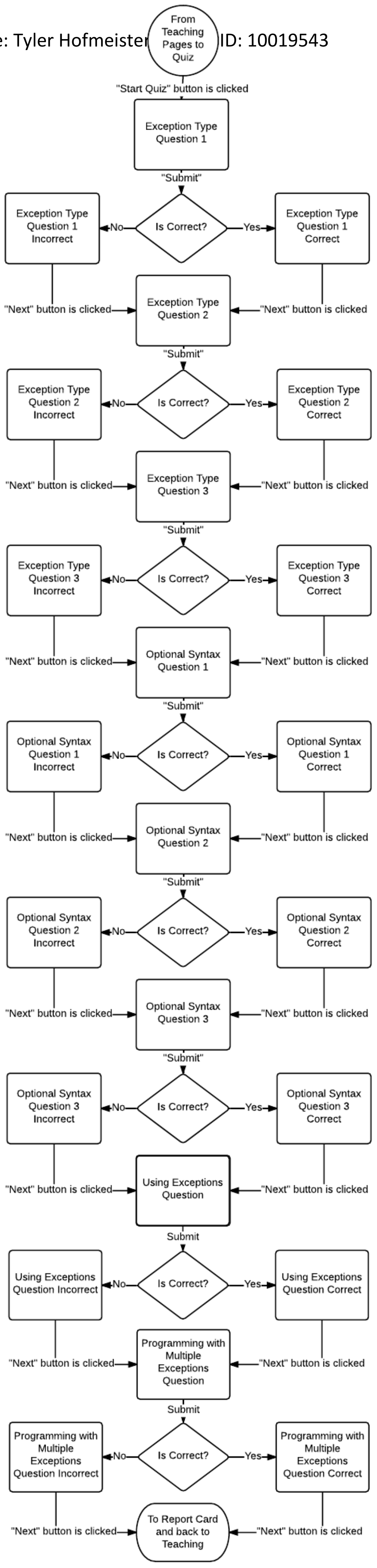

From Login to Teaching Pages
If not Admin and (name is correct or IDNumber does not exist)
Login Information
If the user doesn't exist
If "Report Card" checkbox was checked
Report Card
If the user IDNumber exists and the submitted Name is correct(but not admin)
check the users last current page and return them to that
Make a new user object with the submtted IDNumber and
What is an Error?
"Next" button is clicked
Exceptions and Exception Syntax
"Next" button is clicked
Catching Multiple Exception Types
"Next" button is clicked
Raise, Else, and Finally Statements
Sidebar Navigation
If Error is generated
To Generic Error Page
Exceptions as Variables
"Next" button is clicked
Using Exceptions
"Next" button is clicked
Additional Logic vs. Exceptions
"Next" button is clicked
Sources
"Next" button is clicked
Quiz Introduction Page
"Start Quiz" button is clicked
From Teaching Pages to Quiz

The Testing Pages

The testing pages follow a linear format. The user answers a series of questions and, depending on their answer to each, is either directed towards a page informing them that they were correct or a page informing them that their submitted answer was incorrect. Pages that inform the user that they submitted a wrong answer include a simple animation that shows them the correct answer if they so chose.

There are four different "sections" in the testing part of the website. The first tests the users ability to predict which specific exception types will be generated by presented code. These questions are answered by entering text into a field and submitting the answer to the website's backend. The second section tests the users understanding of how some of the optional syntax that can be used in try statements works. Again, a user must predict the output of presented programs. However, the programs contain try statements with raise, else, and finally statements. These questions are multiple choice; the user must select the answer from a drop-down menu and submit them to the system. The third section, also multiple choice, tests if the user understands how to use try statements in python to actually solve the underling problem(instead of just printing an error message).. The fourth and final section tests the users understanding of how to program with multiple exception types. In order to answer this question, the user must upload a python program that the system will judge.

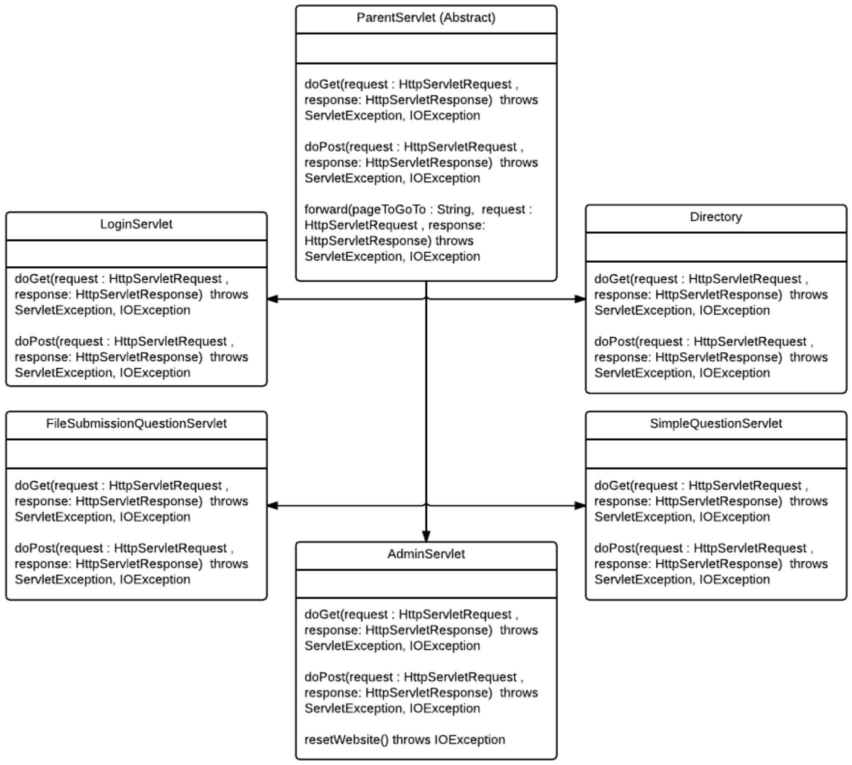
It is worth noting that when a user, while within the testing pages of the website, exits the website and logs back in with the same IDNumber and name, they will be returned to the page they exited from. It is also worth noting that a user who tries to attempt a question more than once will be directed to an error page. A user may only access the testing pages if they have not already completed the quiz. Upon completion, the user will be directed to the report card page where their progress is displayed.


From Teaching Pages to Quiz
"Start Quiz" button is clicked
Exception Type Question 1
"Submit"
Is Correct?
No
Yes
Exception Type Question 1 Incorrect
Exception Type Question 1 Correct
"Next" button is clicked
Exception Type Question 2
Exception Type Question 2 Incorrect
Exception Type Question 2 Correct
Exception Type Question 3
Exception Type Question 3 Incorrect
Exception Type Question 3 Correct
Optional Syntax Question 1
Optional Syntax Question 1 Incorrect
Optional Syntax Question 1 Correct
Optional Syntax Question 2
Optional Syntax Question 2 Incorrect
Optional Syntax Question 2 Correct
Optional Syntax Question 3
Optional Syntax Question 3 Incorrect
Optional Syntax Question 3 Correct
Using Exceptions Question
Submit
Using Exceptions Question Incorrect
Using Exceptions Question Correct
Programming with Multiple Exceptions Question
Programming with Multiple Exceptions Question Incorrect
Programming with Multiple Exceptions Question Correct
To Report Card and back to Teaching

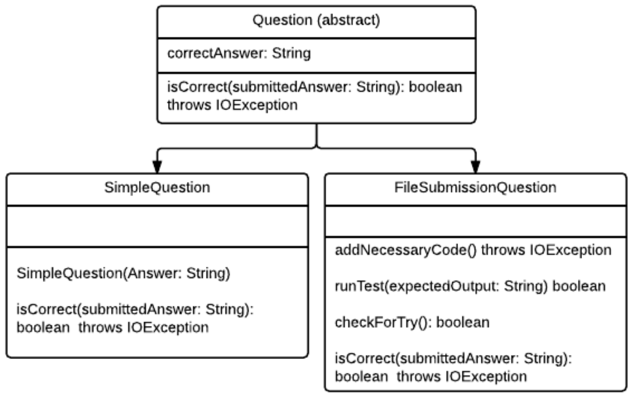
Inheritance between Classes

The following UML diagrams demonstrate the inheritance between various the various classes used in this project. The dependencies between classes will be shown in a later section.

All servlets are subclasses of the abstract ParentServlet class. This is so that each servlet can inherit the forward method, which is uses the server to direct the user to other pages in the website.

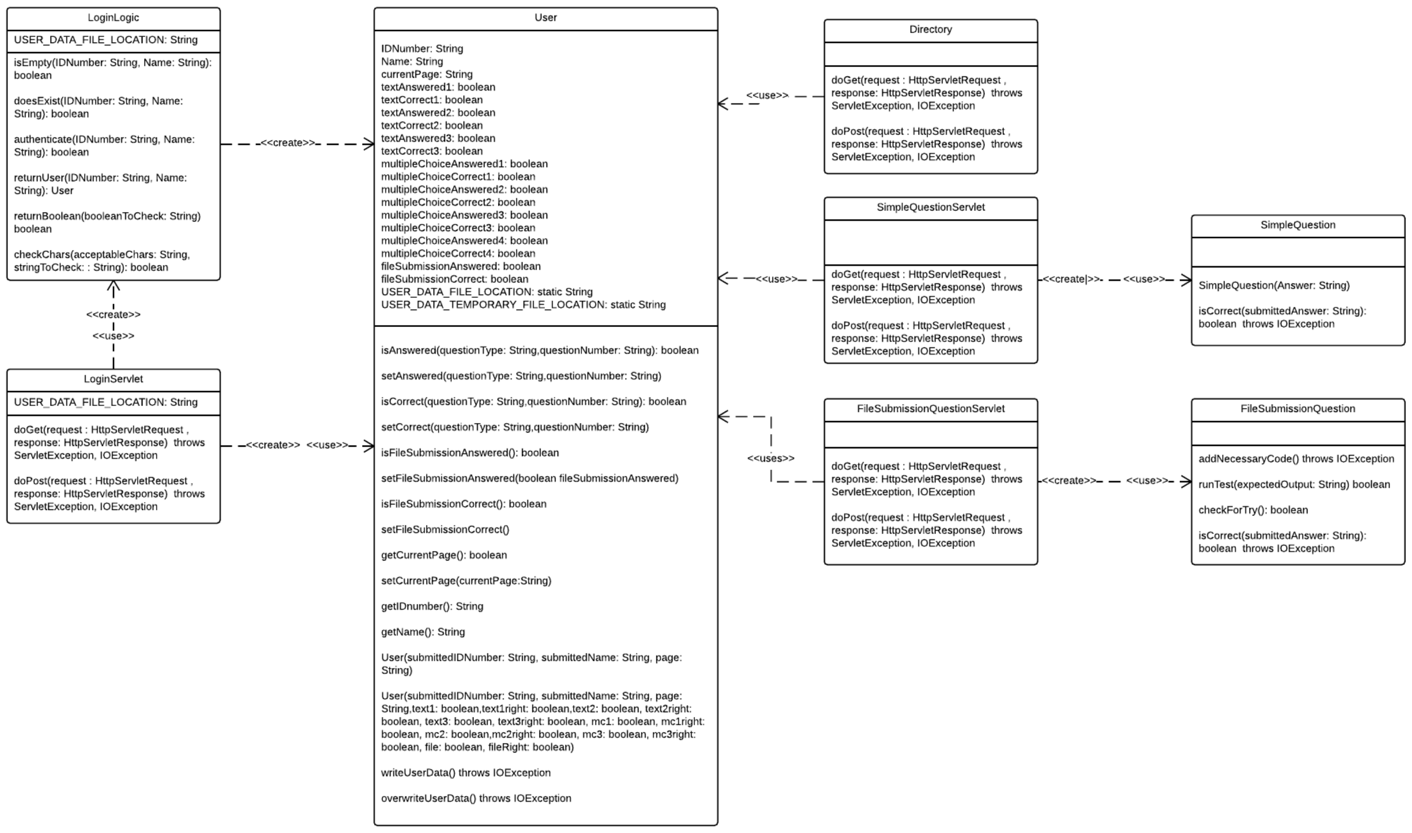

ParentServlet (Abstract)
doGet(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
doPost(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
forward(pageToGoTo : String, request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
LoginServlet
Directory
FileSubmissionQuestionServlet
SimpleQuestionServlet
AdminServlet
resetWebsite() throws IOException

Similarly, the question abstract class is a super class of both FileSubmissionQuestion.class and SimpleQuestion.class. They inherit the correctAnswer instance variable and the isCorrect method. These methods are described in a later section of the document.


Question (abstract)
correctAnswer: String
isCorrect(submittedAnswer: String): boolean throws IOException
SimpleQuestion
SimpleQuestion(Answer: String)
isCorrect(submittedAnswer: String): boolean throws IOException
FileSubmissionQuestion
addNecessaryCode() throws IOException
runTest(expectedOutput: String) boolean
checkForTry(): boolean
isCorrect(submittedAnswer: String): boolean throws IOException

Dependencies between Classes

The following UML diagrams demonstrate the dependencies between various the various classes used in the website.


LoginLogic
USER_DATA_FILE_LOCATION: String
isEmpty(IDNumber: String, Name: String): boolean
doesExist(IDNumber: String, Name: String): boolean
authenticate(IDNumber: String, Name: String): boolean
returnUser(IDNumber: String, Name: String): User
returnBoolean(booleanToCheck: String) boolean
checkChars(acceptableChars: String, stringToCheck: : String): boolean
<<create>>
<<create>>
<<use>>
LoginServlet
USER_DATA_FILE_LOCATION: String
doGet(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
doPost(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
<<create>> <<use>>
User
IDNumber: String
Name: String
currentPage: String
textAnswered1: boolean
textCorrect1: boolean
textAnswered2: boolean
textCorrect2: boolean
textAnswered3: boolean
textCorrect3: boolean
multipleChoiceAnswered1: boolean
multipleChoiceCorrect1: boolean
multipleChoiceAnswered2: boolean
multipleChoiceCorrect2: boolean
multipleChoiceAnswered3: boolean
multipleChoiceCorrect3: boolean
multipleChoiceAnswered4: boolean
multipleChoiceCorrect4: boolean
fileSubmissionAnswered: boolean
fileSubmissionCorrect: boolean
USER_DATA_FILE_LOCATION: static String
USER_DATA_TEMPORARY_FILE_LOCATION: static String
isAnswered(questionType: String,questionNumber: String): boolean
setAnswered(questionType: String,questionNumber: String)
isCorrect(questionType: String,questionNumber: String): boolean
setCorrect(questionType: String,questionNumber: String)
isFileSubmissionAnswered(): boolean
setFileSubmissionAnswered(boolean fileSubmissionAnswered)
isFileSubmissionCorrect(): boolean
setFileSubmissionCorrect()
getCurrentPage(): boolean
setCurrentPage(currentPage:String)
getIDnumber(): String
getName(): String
User(submittedIDNumber: String, submittedName: String, page: String)
User(submittedIDNumber: String, submittedName: String, page: String,text1: boolean,text1right: boolean,text2: boolean, text2right: boolean, text3: boolean, text3right: boolean, mc1: boolean, mc1right: boolean, mc2: boolean,mc2right: boolean, mc3: boolean, mc3right: boolean, file: boolean, fileRight: boolean)
writeUserData() throws IOException
overwriteUserData() throws IOException
<<use>>
Directory
doGet(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
doPost(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
<<use>>
SimpleQuestionServlet
doGet(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
doPost(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
<<create|>> <<use>>
SimpleQuestion
SimpleQuestion(Answer: String)
isCorrect(submittedAnswer: String): boolean throws IOException
<<uses>>
FileSubmissionQuestionServlet
doGet(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
doPost(request : HttpServletRequest , response: HttpServletResponse) throws ServletException, IOException
<<create>> <<use>>
FileSubmissionQuestion
addNecessaryCode() throws IOException
runTest(expectedOutput: String) boolean
checkForTry(): boolean
isCorrect(submittedAnswer: String): boolean throws IOException

The LoginServlet creates an instance of the LoginLogic class and uses this class to test the submitted parameters from the user during login. If the user has submitted an IDNumber and Name that have already been used, the LoginLogic class creates an instance of the user class. If the IDNumber is new, the LoginServlet creates an instance of the User class. The LoginServlet then uses the User class to change the text file which stores data about all existing users.

The Directory, SimpleQuestionServlet, and FileSubmissionQuestionServlet all use the User class to update the stored information regarding the user currently using the website. SimpleQuestionServlet creates and instance of SimpleQuestion and uses it to judge the correctness of submitted answers. FileSubmissionQuestionServlet does the same with the FileSubmissionQuestion class.

Classes and Methods used on the Server:

User.class

This class contains all information the site needs regarding each individual user, is serialized, and should only have 1 active per session.

Variables:

- boolean instance variables to show that the user has answered each question on the website; a complete list is of these variables is available in the UML diagram detailing this class.
- string instance variables for IDnumber, name, and currentPage
- static string class variables that hold the path to the text file that stores all user data, and a temporary text file which is used when re-writing the original file.

Methods:

- isFileSubmissionAnswered(): this method returns the boolean instance variable indicating whether or not the user has answered the file submission question.
- setFileSubmissionAnswered(fileSubmissionAnswered: boolean): this method changes the value of the fileSubmissionAnswered boolean instance variable, based on the boolean argument variable.
- isFileSubmissionCorrect(): this method returns the boolean instance variable indicating whether or not the user has answered the file submission question correctly.
- setFileSubmissionCorrect(fileSubmissionCorrect: boolean): this method changes the value of the fileSubmissionCorrect boolean instance variable, based on the boolean argument variable.
- isAnswered(questionType: String, questionNumber: String): this method checks the value of the boolean instance variable indicating that the question with submitted question type and number has been answered or not.
- setAnswered(questionType: String, questionNumber: String): this method changes the value of the boolean instance variable indicating that the question with submitted question type and number has been answered or not.
- isCorrect(questionType: String, questionNumber: String): this method checks the value of the boolean instance variable indicating that the question with submitted question type and number has been answered correctly or not.
- setCorrect(questionType: String, questionNumber: String): this method changes the value of the boolean instance variable indicating that the question with submitted question type and number has been answered correctly or not.
- getCurrentPage(): returns the current page string variable of the User that calls the method.
- setCurrentPage(currentPage: String): sets the current page string variable of the User that calls the method, based on the submitted argument.

- getIDNumber(): returns the users string IDNumber variable.
- getName(): returns the users string Name variable.
- User(submittedIDNumber: String, submittedName: String, page: String): creates a new User instance, with the submitted IDNumber, Name, and currentPage. All other instance variables are set as false when using this constructor.
- User(submittedIDNumber: String, submittedName: String, page: String,text1: boolean,text1right: boolean,text2: boolean, text2right: boolean, text3: boolean, text3right: boolean, mc1: boolean, mc1right: boolean, mc2: boolean,mc2right: boolean, mc3: boolean, mc3right: boolean, file: boolean, fileRight: boolean): creates a new User instance, using all of the submitted values for each corresponding instance variable. This constructor is used to create a new User with all of the variable values of a user who has previously used the website.
- writeUserData(): this method writes all of the values for each instance variable of the User to a text file, where the values are parsed with a "/" character.
- overwriteUserData(): this method first creates a new temporary text file and then goes line by line through the existing text file. If a line represents the current user, it instead writes the values supplied by the existing User class. If a line represents another use, it writes that line to the new text file. Once this method has gone through the entire text file, it deletes the old one and replaces it with the new text file which it has created.

<u>LoginLogic.class</u>

Variables:

- static string class variables that hold the path to the text file that stores all user data.

Methods:

- doesExist(IDNumber: String, Name: String): checks to see if the submitted IDNumber exists in the UserData.txt file. Returns a boolean indicating this information.
- Authenticate(IDNumber: String, Name: String): checks to see if the submitted IDNumber and submitted Name strings match the corresponding values in the UserData.txt file. Returns a boolean indicating this information.
- returnUser(IDNumber: String, Name: String): finds the correct User information in the UserData.txt file, and calls the User constructor that can assign values to every instance variable in the User.class in order to create a user identical to the one with IDNumber and Name that was submitted to the system.
- returnBoolean(booleanToCheck: String): takes the submitted booleanToCheck argument and returns the boolean "true" if this string is equal to the string "true" and false otherwise.
- checkChars(string ToCheck: String): checks the submitted string to see if it contains a "/" or "\" character. Returns true if it does, and false otherwise.

Question.class (abstract)

Variables:

- String variable indicating the correct answer.

Methods:

- isCorrect(submittedAnswer: String): (abstract method)

SimpleQuestion.class

Variables:

- (Inherited) string variable indicating the correct answer

Methods:

- isCorrect(submittedAnswer: String): checks to see if the submitted answer string is equal to this instance of the class' correct answer variable. Returns true if they are equal, returns false otherwise.

FileSubmissionQuestion.class

Variables:

- (Inherited) string variable indicating the correct answer

Methods:

- runTest(expectedOutput: String, submitted_expense_report: String, budget_allocated_for_expenses: String): accesses the correct directory in which to run each tests, creates a new processbuilder, starts the process, using the submitted_expense_report and budget_allocated_for_expenses strings as arguments for the command line. This method then sets up a buffered reader, and gives it a finite time to indicate that it is ready to read the output of the command line. If the buffered reader goes over the time limit, this method returns false. If the buffered reader is ready within the time limit, it reads the output of the command line. This output is compared to the expectedOutput. If they are equivalent, this method returns true and false otherwise.
- addNecessaryCode(): adds necessary code so that the runTest method can supply variables via the command line that will be used in the code to generate the expected output.
- checkForTry(): checks the submitted program, line by line, for the string "try". If the program contains this string, this method returns true, and false otherwise.
- isCorrect(): runs tests on the submitted program using the runTest method, the checkForTry() method, and the addNecessaryCode() method in order to see if the submitted program fits all of the required specifications and generates the desired outputs for each test. Returns true if the program passes each test, and returns false if the program fails one of them.

ParentServlet.class (abstract)

- doGet(request: HTTPServletRequest, response: HTTPServletResponse) (abstract)
- doPost(request: HTTPServletRequest, response: HTTPServletResponse) (abstract)
- forward(pageToGoTo: String, request: HTTPServletRequest, response: HTTPServletResponse): gets the servlet context, uses the servlet contet to instantiate a requestdispatcher, and then uses this request dispatcher to direct the user to a specific page using the server and not the user's browser.

Methods:

- LoginServlet(): the automatically generated constructor, inherited from it's super class.
- doGet(request: HTTPServletRequest, response: HTTPServletResponse) calls the doPost method
- doPost(request: HTTPServletRequest, response: HTTPServletResponse): this method gets the parameters submitted from the login page, and assigns them to corresponding variables for IDNumber and Name. First it instantiates the LoginLogic.class in order to call it's methods on the submitted fields. Then, it calls the checkChars method to ensure that the submitted fields don't contain illegal characters ("/" and "\"). If they do, the servlet directs the user to a specific error page informing the user that they used illegal characters. If not, this method checks to see if the submitted fields are "000666" for IDNumber and "super" for Name. If so, it directs the user to the admin page. If not, it checks to see if any of the fields are empty and, if so, directs the user to a specific error page informing the user that they submitted empty fields. If not, it checks to see if the IDNumber exists(using the doesExist method), or is 000666, and if so it calls the authenticate method in order to determine if the submittedName matches this IDNumber. If not, it directs the user to a page informing them of this specific error. If the IDNumber matches it's entered name in UserData.txt, this method call the returnUser method, assigns the recreated User object to the "User" attribute associated with the session and directs them to whatever page they were at last. If the IDNumber does not exist, and is not 000666, then a new User object is created with the submitted parameters, this user object is associated with the running session, and then the user is directed to the "What is an Error" page.

SimpleQuestionServlet.class

Methods:

- SimpleQuestionServlet(): the automatically generated constructor, inherited from it's super class.
- doGet(request: HTTPServletRequest, response: HTTPServletResponse) calls the doPost method
- doPost(request: HTTPServletRequest, response: HTTPServletResponse): gets the User object from the session and then receives parameters for the questionType, questionNumber, submittedAnswer, and correctAnswer from the page calling this servlet. This method then instantiates the SimpleQuestion class with the correctAnswer received from the page assigned to the corresponding variable of this object. This method then calls the IsCorrect method of the SimpleQuestion object to determine if the submittedAnswer is correct. If so, this servlet directs the user to the page indicating they got the question right. If not, this servlet directs the user to the page indicating they got the question wrong. If the user has already answered this question, this method directs them to a generic error page.

<u>FileSubmissionQuestionServlet.class</u>

Methods:

- AdminServlet(): the automatically generated constructor, inherited from it's super class.
- doGet(request: HTTPServletRequest, response: HTTPServletResponse) calls the doPost method
- doPost(request: HTTPServletRequest, response: HTTPServletResponse): gets the User.class object associated with the running session. This method then checks to see if the user has previously answered this question. If so, it directs the user to a generic error page. If not, this method then uses the importedapache.commons.fileupload classes FileItem, DisFileItemFactory, and ServletFileUpload, to take a file, submitted from the user and upload to the correct directory. Note that these classes take care of all the low level processes that must occur in order to upload a file. This method then instantiates the FileSubmissionQuestionServlet, and calls its IsCorrect method in order to determine if the submitted program fulfills the required specifications. If so, it directs the user to a page informing them that they have answered the question correctly. If not, it directs the user to a page informing them that they answred incorrectly.

<u>AdminServlet.class</u>

Variables:

- A static string variable indicating the path to the UserData.txt file.

Methods:

- AdminServlet(): the automatically generated constructor, inherited from it's super class.
- doGet(request: HTTPServletRequest, response: HTTPServletResponse) calls the doPost method
- doPost(request: HTTPServletRequest, response: HTTPServletResponse): with all of it's code surrounded in a try, catch statement, this method attempts to call the resetWebsite method and if this method is successful, it directs the user to the "Confirm Restart" web page. If the resetWebsite method fails, or an exception is raised, it directs the user to a generic Error Page.
- resetWebsite(): this file deletes the current UserData.txt file and creates a new one with the same filepath.

<u>Directory.class</u>

Methods:

- Directory(): the automatically generated constructor, inherited from it's super class.
- doGet(request: HTTPServletRequest, response: HTTPServletResponse) calls the doPost method
- doPost(request: HTTPServletRequest, response: HTTPServletResponse): checks to see if the user is being directed by a webpage to the login page, if so, directs them there immediately. Otherwise, this method gets the session of the user, and from the session gets the attribute labelled "User" that contains the instantiated User class associated with the running session. This method then changes that User's currentPage variable to be the page that the user is being directed to, and then directs them there.

Comments on the 3 parts of the Site:

The login part of the website was designed in order to thoroughly test the submitted parameters to ensure that they wouldn't cause any the website to malfunction. Blank fields and fields containing slashes direct the user to the error page because these fields could potentially cause problems when saving User data and retrieving user data. While it wasn't included in the given specifications, I wanted to have the user be directed back to the page they left from on the website. This would ensure that a user couldn't cheat on the quiz by accessing the teaching pages inbetween questions. It also provides convenience for the user. While originally the website only directed the user to a generic error page, I was informed that multiple error pages explaining to the user what specifically they did wrong would be helpful so I was glad to add them. This was something that I had not considered before the meeting; if I had I would have likely added this feature to other parts of the website.

The teaching pages were designed to be as thorough as humanly possible. Several textbooks were consulted along with the standard online python references in order to ensure I didn't leave any concept regarding Exceptions and Exception handling untouched. The caveat of having so much information on the teaching pages was the need to have additional questions in order to test the user on more material. I tried to give the user multiple ways of browsing the information; with both a sidebar navigation and "next" buttons so that they could navigate the website in a linear fashion. Having animations display the output of various program in the teaching pages was something that I originally wanted to include, as this always helped me better understand syntax and the use of various constructs in a programming language.

As mentioned, the thorough teaching pages prompted the testing pages to also be more thorough. I tried to include a variety of questions that covered all of the material taught in the website. Judging the file submission to be correct was an interesting problem that I devoted a great deal of time. I spent many hours to find unconventional means of rendering the tests ineffective. Testing for infinite loops or having the program wait for input was an idea I am particularly proud of as this caused my website to timeout instead of generating an error page.