Part 5 : Final Report

Steven Jace Conflenti, Jennifer Michael, & Cameron Taylor

1. List the features that were implemented

ID	Feature
UR-01	As a user, I need to be able to select a default set of terms and then play the game.
UR-02	As a user, I need to be able to create by own set of terms and values to be be tested against in the game.
UR-03	As a user, I need to be able to import a set of terms and values from a text file to be tested against in the game.
UR-04	As a user, I need to be able to edit or delete a set of key terms and values after initially creating it.
UR-05	As a user, I need to be able to view my scores for different sets of test terms.
UR-06	As a user, I need to be able to export a set of key terms and values to a text file.
UR-07	As a user, I need to be able to type the corresponding value of a key term in the game in order to earn points.
UR-08	As a user, I can specify the difficulty level of the game.
UR-09	As a user, I want to be able to see how I’ve improved over time while playing with my sets by viewing a scoreboard.
UR-10	As a user, I want to be able to quit the game from the Home page.
FR-01	When a user types the value of a key term displayed in the game, the score needs to be updated and the asteroid needs to be destroyed.
FR-02	The difficulty level must be proportional to the amount of time the user has to type the value for a given term.
FR-05	When a user enters the value of a key term it needs to be validated against the correct value, disregarding differences in punctuation such as apostrophes, dashes, commas, and periods as well as disregarding differences in capitalization.
NFR-01	When a user selects a set to play with, all of the keys and values from the set must be loaded in from the database.

NFR-04	The game must run in a Linux environment.
NFR-05	Terms and values entered by the user must be checked to ensure user input doesn’t compromise database integrity.

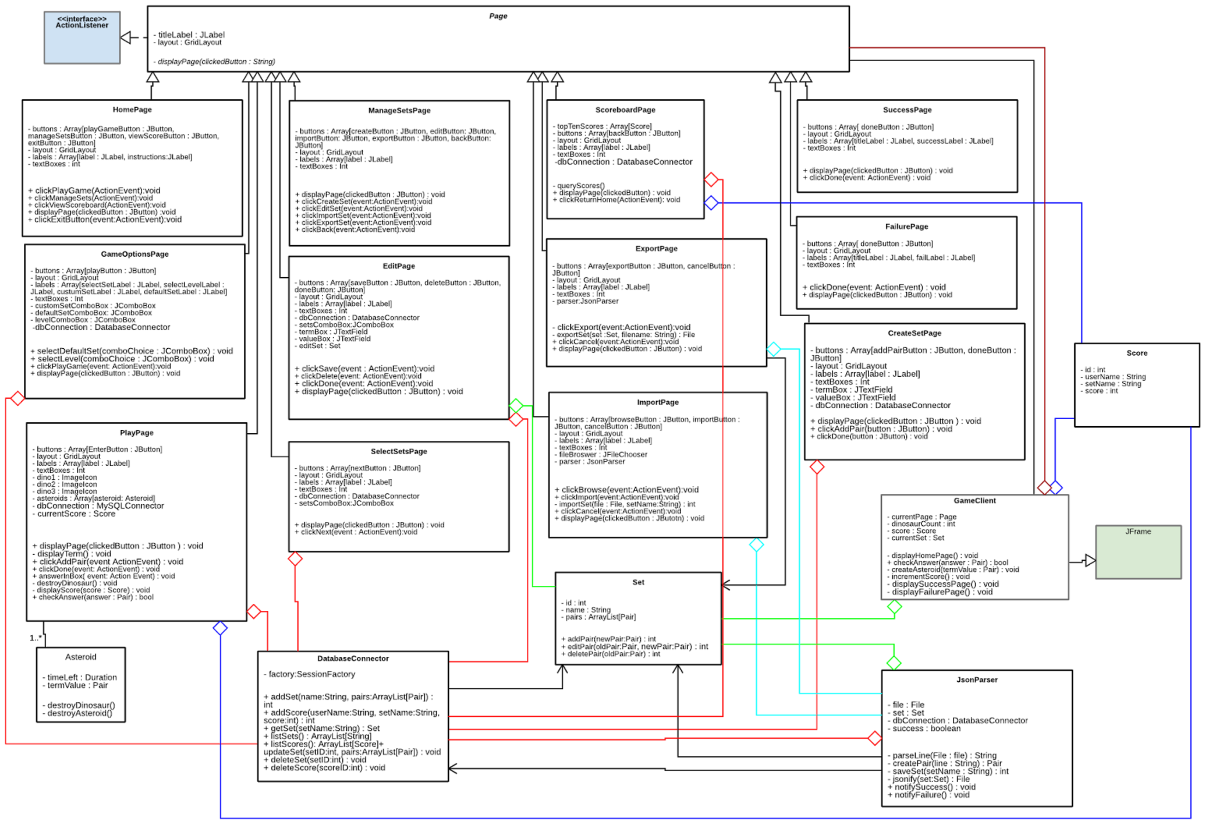
2. List the features that were not implemented

We did not have a chance to implement the dinosaur/asteroid graphic into our game. Everything functions as originally planned, but requirements pertaining to “dinosaurs disappearing” or “asteroids falling” were not met for this reason. In this table, we have commented how the functionality was implemented without the graphical features.

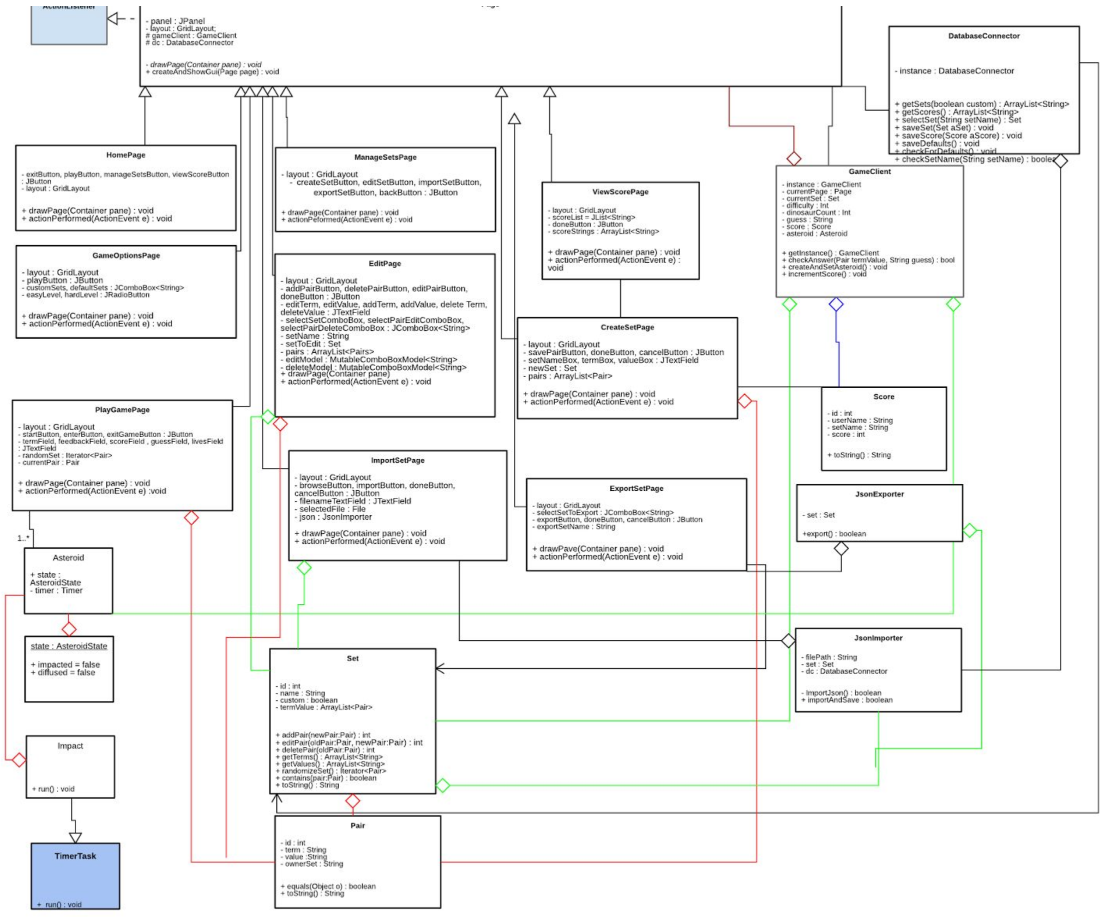
ID	Feature	Comments
FR-03	When a term falls off the screen, a dinosaur must disappear from the screen as well.	We use a timer-- the the user does not answer within the allotted amount of time, they lose a life.
FR-04	When all dinosaurs are no longer on the bottom of the screen, the game must end and display the score to the user.	When all lives are lost, the game ends.
FR-05	When all terms have been asked and there is at least one dinosaur remaining, the game must end and display the score to the user.	When all terms have been asked and not all lives have been used, the game ends.
NFR-02	Game play will be easy to learn because instructions will be available on the Home screen.	We did not feel it was necessary as the game play objective was easy and instructions would have cluttered the Home Page.
NFR-03	The asteroid containing a key term must disappear within one second after it’s value is correctly entered by the user.	There were not asteroids to display, but a new term is displayed within one second after a user enters a correct value.

3. Show Part 2 class diagram and your final class diagram. What changed? Why?

Old Diagram:


<<interface>>
ActionListener
Page
- titleLabel : JLabel
- layout : GridLayout
- displayPage(clickedButton : String)
HomePage
- buttons : Array[playGameButton : JButton, manageSetsButton : JButton, viewScoreButton : JButton, exitButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel, instructions:JLabel]
- textBoxes : int
+ clickPlayGame(ActionEvent):void
+ clickManageSets(ActionEvent):void
+ clickViewScoreboard(ActionEvent):void
+ displayPage(clickedButton : JButton) :void
+ clickExitButton(event:ActionEvent):void
ManageSetsPage
- buttons : Array[createButton : JButton, editButton: JButton, importButton: JButton, exportButton : JButton, backButton: JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
+ displayPage(clickedButton : JButton) : void
+ clickCreateSet(event:ActionEvent):void
+ clickEditSet(event:ActionEvent):void
+ clickImportSet(event:ActionEvent):void
+ clickExportSet(event:ActionEvent):void
+ clickBack(event:ActionEvent):void
ScoreboardPage
- topTenScores : Array[Score]
- buttons : Array[backButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
-dbConnection : DatabaseConnector
- queryScores()
+ displayPage(clickedButton) : void
+ clickReturnHome(ActionEvent): void
SuccessPage
- buttons : Array[doneButton : JButton]
- layout : GridLayout
- labels : Array[titleLabel : JLabel, successLabel : JLabel]
- textBoxes : Int
+ displayPage(clickedButton : JButton) : void
+ clickDone(event: ActionEvent) : void
FailurePage
- buttons : Array[doneButton : JButton]
- layout : GridLayout
- labels : Array[titleLabel : JLabel, failLabel : JLabel]
- textBoxes : Int
+ clickDone(event: ActionEvent) : void
+ displayPage(clickedButton : JButton) : void
GameOptionsPage
- buttons : Array[playButton : JButton]
- layout : GridLayout
- labels : Array[selectSetLabel : JLabel, selectLevelLabel : JLabel, custumSetLabel : JLabel, defaultSetLabel : JLabel]
- textBoxes : Int
- customSetComboBox : JComboBox
- defaultSetComboBox: JComboBox
- levelComboBox : JComboBox
-dbConnection : DatabaseConnector
+ selectDefaultSet(comboChoice : JComboBox) : void
+ selectLevel(comboChoice : JComboBox) : void
+ clickPlayGame(event: ActionEvent):void
+ displayPage(clickedButton : JButton) : void
EditPage
- buttons : Array[saveButton : JButton, deleteButton : JButton, doneButton: JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- dbConnection : DatabaseConnector
- setsComboBox:JComboBox
- termBox : JTextField
- valueBox : JTextField
- editSet : Set
+ clickSave(event : ActionEvent):void
+ clickDelete(event: ActionEvent):void
+ clickDone(event: ActionEvent):void
+ displayPage(clickedButton : JButton) : void
ExportPage
- buttons : Array[exportButton : JButton, cancelButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- parser:JsonParser
- clickExport(event:ActionEvent):void
- exportSet(set :Set, filename: String) : File
+ clickCancel(event:ActionEvent):void
+ displayPage(clickedButton : JButton) : void
CreateSetPage
- buttons : Array[addPairButton : JButton, doneButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- termBox : JTextField
- valueBox : JTextField
- dbConnection : DatabaseConnector
+ displayPage(clickedButton : JButton) : void
+ clickAddPair(button : JButton) : void
+ clickDone(button : JButton) : void
Score
- id : int
- userName : String
- setName : String
- score : int
PlayPage
- buttons : Array[EnterButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- dino1 : ImageIcon
- dino2 : ImageIcon
- dino3 : ImageIcon
- asteroids : Array[asteroid: Asteroid]
- dbConnection : MySQLConnector
- currentScore : Score
+ displayPage(clickedButton : JButton) : void
- displayTerm() : void
+ clickAddPair(event ActionEvent) : void
+ clickDone(event: ActionEvent) : void
+ answerInBox(event: Action Event) : void
- destroyDinosaur() : void
- displayScore(score : Score) : void
+ checkAnswer(answer : Pair) : bool
SelectSetsPage
- buttons : Array[nextButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- dbConnection : DatabaseConnector
- setsComboBox:JComboBox
+ displayPage(clickedButton : JButton) : void
+ clickNext(event : ActionEvent):void
ImportPage
- buttons : Array[browseButton : JButton, importButton : JButton, cancelButton : JButton]
- layout : GridLayout
- labels : Array[label : JLabel]
- textBoxes : Int
- fileBroswer : JFileChooser
- parser : JsonParser
+ clickBrowse(event:ActionEvent):void
+ clickImport(event:ActionEvent):void
- importSet(file : File, setName:String) : int
+ clickCancel(event:ActionEvent):void
+ displayPage(clickedButton : JButotn) : void
GameClient
- currentPage : Page
- dinosaurCount : int
- score : Score
- currentSet : Set
- displayHomePage() : void
+ checkAnswer(answer : Pair) : bool
- createAsteroid(termValue : Pair) : void
- incrementScore() : void
- displaySuccessPage() : void
- displayFailurePage() : void
JFrame
1..*
Asteroid
- timeLeft : Duration
- termValue : Pair
- destroyDinosaur()
- destroyAsteroid()
Set
- id : int
- name : String
- pairs : ArrayList[Pair]
+ addPair(newPair:Pair) : int
+ editPair(oldPair:Pair, newPair:Pair) : int
+ deletePair(oldPair:Pair) : int
DatabaseConnector
- factory:SessionFactory
+ addSet(name:String, pairs:ArrayList[Pair]) : int
+ addScore(userName:String, setName:String, score:int) : int
+ getSet(setName:String) : Set
+ listSets() : ArrayList[String]
+ listScores(): ArrayList[Score]+
updateSet(setID:int, pairs:ArrayList[Pair]) : void
+ deleteSet(setID:int) : void
+ deleteScore(scoreID:int) : void
JsonParser
- file : File
- set : Set
- dbConnection : DatabaseConnector
- success : boolean
- parseLine(File : file) : String
- createPair(line : String) : Pair
- saveSet(setName : String) : int
- jsonify(set:Set) : File
+ notifySuccess() : void
+ notifyFailure() : void

New Diagram:


- panel : JPanel
- layout : GridLayout;
# gameClient : GameClient
# dc : DatabaseConnector
- drawPage(Container pane) : void
+ createAndShowGui(Page page) : void
HomePage
- exitButton, playButton, manageSetsButton, viewScoreButton : JButton
- layout : GridLayout
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) : void
GameOptionsPage
- layout : GridLayout
- playButton : JButton
- customSets, defaultSets : JComboBox<String>
- easyLevel, hardLevel : JRadioButton
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) : void
ManageSetsPage
- layout : GridLayout
- createSetButton, editSetButton, importSetButton, exportSetButton, backButton : JButton
+ drawPage(Container pane) : void
+ actionPeformed(ActionEvent e) : void
EditPage
- layout : GridLayout
- addPairButton, deletePairButton, editPairButton, doneButton : JButton
- editTerm, editValue, addTerm, addValue, delete Term, deleteValue : JTextField
- selectSetComboBox, selectPairEditComboBox, selectPairDeleteComboBox : JComboBox<String>
- setName : String
- setToEdit : Set
- pairs : ArrayList<Pairs>
- editModel : MutableComboBoxModel<String>
- deleteModel : MutableComboBoxModel<String>
+ drawPage(Container pane)
+ actionPerformed(ActionEvent e) : void
ViewScorePage
- layout : GridLayout
- scoreList = JList<String>
- doneButton : JButton
- scoreStrings : ArrayList<String>
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) : void
DatabaseConnector
- instance : DatabaseConnector
+ getSets(boolean custom) : ArrayList<String>
+ getScores() : ArrayList<String>
+ selectSet(String setName) : Set
+ saveSet(Set aSet) : void
+ saveScore(Score aScore) : void
+ saveDefaults() : void
+ checkForDefaults() : void
+ checkSetName(String setName) : boolea
GameClient
- instance : GameClient
- currentPage : Page
- currentSet : Set
- difficulty : Int
- dinosaurCount : Int
- guess : String
- score : Score
- asteroid : Asteroid
+ getInstance() : GameClient
+ checkAnswer(Pair termValue, String guess) : bool
+ createAndSetAsteroid() : void
+ incrementScore() : void
CreateSetPage
- layout : GridLayout
- savePairButton, doneButton, cancelButton : JButton
- setNameBox, termBox, valueBox : JTextField
- newSet : Set
- pairs : ArrayList<Pair>
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) : void
Score
- id : int
- userName : String
- setName : String
- score : int
+ toString() : String
PlayGamePage
- layout : GridLayout
- startButton, enterButton, exitGameButton : JButton
- termField, feedbackField, scoreField , guessField, livesField : JTextField
- randomSet : Iterator<Pair>
- currentPair : Pair
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) :void
ImportSetPage
- layout : GridLayout
- browseButton, importButton, doneButton, cancelButton : JButton
- filenameTextField : JTextField
- selectedFile : File
- json : JsonImporter
+ drawPage(Container pane) : void
+ actionPerformed(ActionEvent e) : void
ExportSetPage
- layout : GridLayout
- selectSetToExport : JComboBox<String>
- exportButton, doneButton, cancelButton : JButton
- exportSetName : String
+ drawPave(Container pane) : void
+ actionPerformed(ActionEvent e) : void
JsonExporter
- set : Set
+export() : boolean
1..*
Asteroid
+ state : AsteroidState
- timer : Timer
state : AsteroidState
+ impacted = false
+ diffused = false
JsonImporter
- filePath : String
- set : Set
- dc : DatabaseConnector
- ImportJson() : boolean
+ importAndSave : boolean
Set
- id : int
- name : String
- custom : boolean
- termValue : ArrayList<Pair>
+ addPair(newPair:Pair) : int
+ editPair(oldPair:Pair, newPair:Pair) : int
+ deletePair(oldPair:Pair) : int
+ getTerms() : ArrayList<String>
+ getValues() : ArrayList<String>
+ randomizeSet() : Iterator<Pair>
+ contains(pair:Pair) : boolean
+ toString() : String
Impact
+ run() : void
Pair
- id : int
- term : String
- value :String
- ownerSet : String
+ equals(Object o) : boolean
+ toString() : String
TimerTask
+ run() : void

Easier to read and zoom in: https://www.gliffy.com/go/publish/11275653

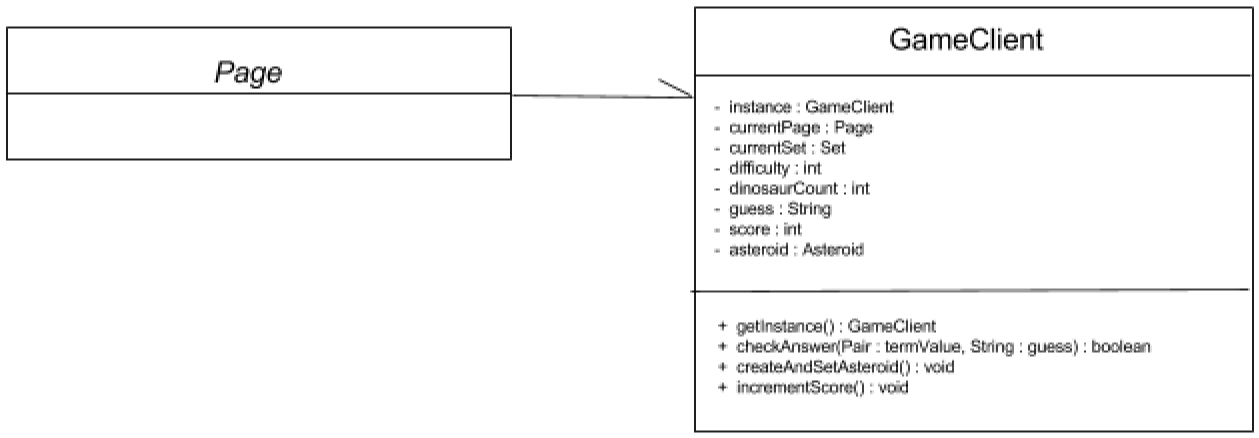
Changes:

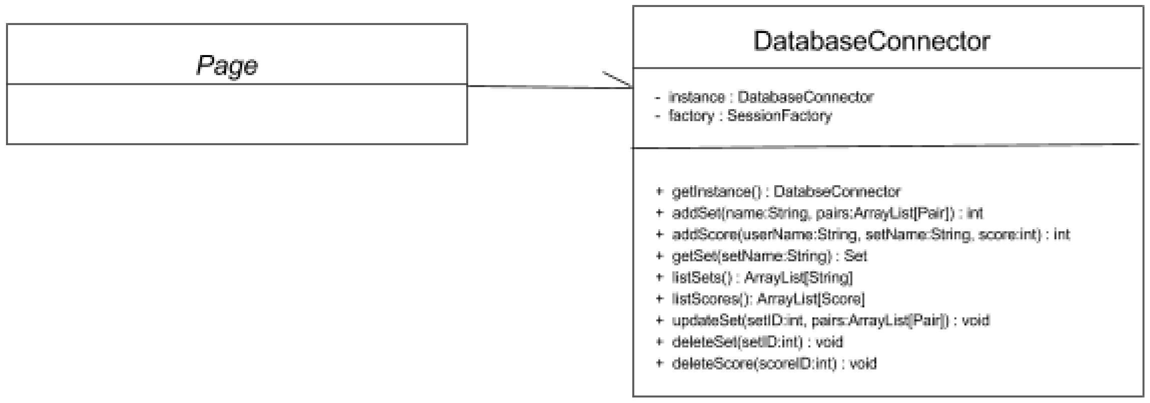
- GameClient and DatabaseConnector became Singleton classes that were instance variables of the abstract Page class.
- JsonParser was split into two classes: JsonImporter and JsonExporter
- We added a Pair class which contains instance variables term, value, and ownerSet. Set objects contain an ArrayList of these Pairs.
- Asteroid implements the Java TimerTask interface
- Page classes only contain two methods: drawPage and actionPerformed instead of separate functions for each action performed. This came from gaining more knowledge about the ActionListener interface and Java Swing.
- We no longer have Failure and Success page classes because they seemed unnecessary.
- Set manipulation page classes make sure of Set and Pair objects.

Besides that, most things are the same or very similar! The changes we had mainly came about after learning more about the Java interfaces we were implementing. We didn’t have the proper knowledge about them when making the class diagram to make things super accurate. Other changes came about because we realized we had planned a redundancy or needed to make classes more encapsulated. Designing up front was very helpful for our application because it gave us a great stepping off point--when we started development, we already knew what we needed to do. We had done so much planning that the implementation came easy and we encountered few architectural problems.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how?

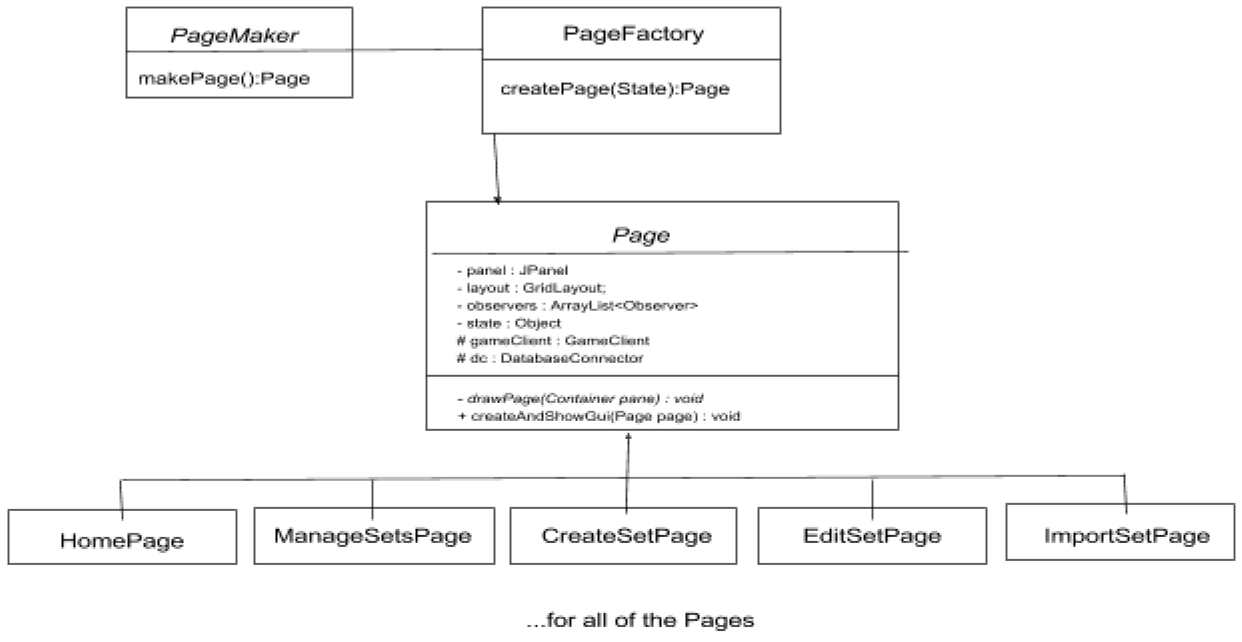
In our final prototype, we made use of the singleton design pattern in both our GameClient and DatabaseConnector classes. We realized that the pattern applied well here because only one instance of each of those objects should exist at a time. If we had multiple GameClient objects, for example, there would be problems associated with which instance’s instance variables to use: which instance’s currentPage should be displayed, which instance’s currentSet should be in play, etc… Having it be a singleton class made that easier. The same applied for the DatabaseConnector class, but it also made things faster since we didn’t have to instantiate such a time-expensive class repeatedly throughout the life of the program.




Page
DatabaseConnector
- instance : DatabaseConnector
- factory : SessionFactory
+ getInstance() : DatabseConnector
+ addSet(name:String, pairs:ArrayList[Pair]) : int
+ addScore(userName:String, setName:String, score:int) : int
+ getSet(setName:String) : Set
+ listSets() : ArrayList[String]
+ listScores(): ArrayList[Score]
+ updateSet(setID:int, pairs:ArrayList[Pair]) : void
+ deleteSet(setID:int) : void
+ deleteScore(scoreID:int) : void

At first, we attempted to use the observer pattern in our implementation, however, we got confused over the architecture, and ran into issues. We thought that observer would be good for communication between GameClient and our Asteroids; however, in our final implementation we realized that only one Asteroid was ever alive at a time. Since this is just basic one-to-one communication that can be achieve through getters and setters we ended up ripping out our observer and subject interfaces mid-way through. However, before we settled on going the getter/setter route we were also thinking that the mediator pattern could be appropriate for what we were originally trying to accomplish with observer. Similarly to observer, we eventually decided not to implement this in our application due to the simplicity of it. As we previously stated, all of the class communication in our app is one-to-one, so we didn't think mediator (which excels in situations where there's a many-to-many communication pattern) would really benefit us. That being said, implementing a pattern like mediator may have allowed us to have looser coupling between our objects which is ultimately what we were hoping to achieve.

Another pattern that would have worked in our project is the factory pattern. We have a bunch of GUI Page classes that are instantiated with a user clicks a button. A Factory class would consolidate all the instantiation of these classes into one area and would take the programmer's job of deciding which class to instantiate away. The Factory class would take as input some sort of state--possibly the currentPage and the button clicked by the user-- and make a decision based off that input which page to instantiate.



5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design, and implement a system?

One thing we learned about the process of analysis and design is that unless you have a lot of experience it is hard to have enough foresight to architect a system from the ground the first time. Going into the design process, we had a good idea of what we wanted our system to do from a high-level, but we lacked knowledge of Java and experience with design patterns. This led us to some decisions we shifted away from as we progressed through our project and learned more in those areas. Simply speaking, we tried to design a system without having knowledge of the full capabilities of the tools we were going to use. One example of this is that going into the project we only had a vague idea of Swing and how to make UI's in java out of the various components and layouts. This led us to making changes to our original designs for our Pages as we learned more about the capabilities we had. Another example is our misapplication of the observer pattern. Over the course of the project we transitioned from not planning on using observer at all, to implementing it and using it incorrectly, and finally to pulling it back out and using something more appropriate. We made this mistake simply because we are inexperienced and had some misunderstanding/confusion relating to the pattern.

Going through this process also forced us to learn the importance and value that refactoring can provide. Refactoring allowed us to actually make the changes we needed to make as worked through our project and learned more as I just described. Although it takes time away from implementing other functionality, refactoring allowed us to incrementally improve our design and the integrity of our code which was (and still is) greatly needed for our project. If we were going to continue working on our project, we would surely continue to refactor our existing code especially by trying to implement more design patterns. We saw the understandability of our code improve immensely as we simplified and refactored it, so we would definitely want to continue this process. Overall, we learned a lot about how the process of analysis and design interacts with the actual implementation process, and how we can nimbly navigate through both of them together.