Kings and Creatures Final Report

Ameya Bahirat, Ben Droste, Peter McPadden

1. List features implemented (table with ID and title)

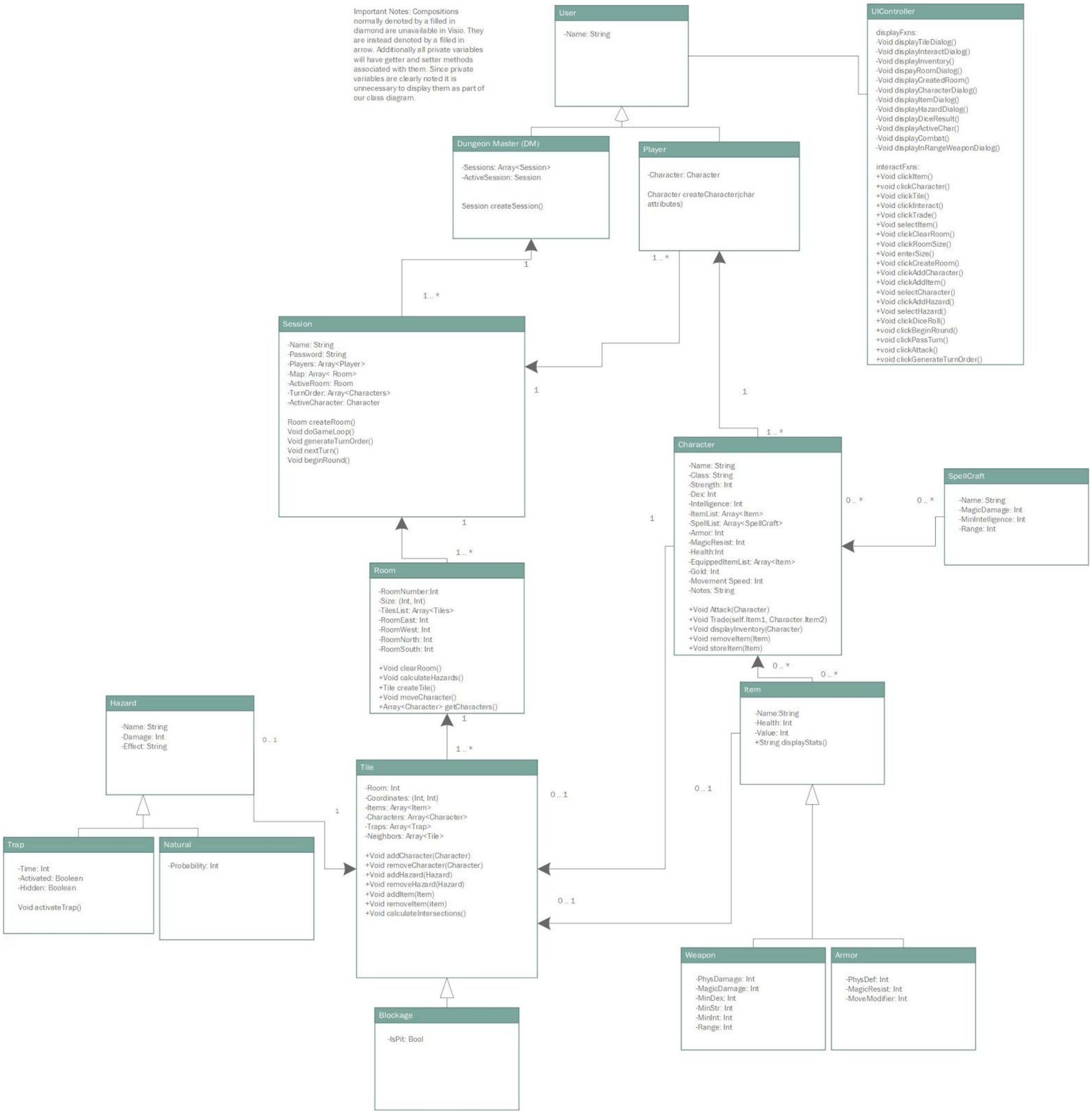
ID	Title
US-006	Initiate Trade with NPC
US-008	Players can create a character
US-002	Create a turn order that can be iterated through
US-004	Players can initiate combat
US-005	The results of combat are visible to players
US-003	NPC’s can attack player characters

2. List the features were not implemented from Part 2 (table with ID and title).

ID	Title
US-001	DM should be able to create a scenario with NPCs, Rooms, and Enemies
US-005	On combats the computer should display to users precalculated success minimum, and current dice roll(s)
US-007	Dungeon Master should be able to generate new room when players decide to move out of the current room
BR-001	If a session is private. A password must be supplied to enter.
BR-002	Users must not be able to use Dungeon Master (DM) controls if they are not the DM

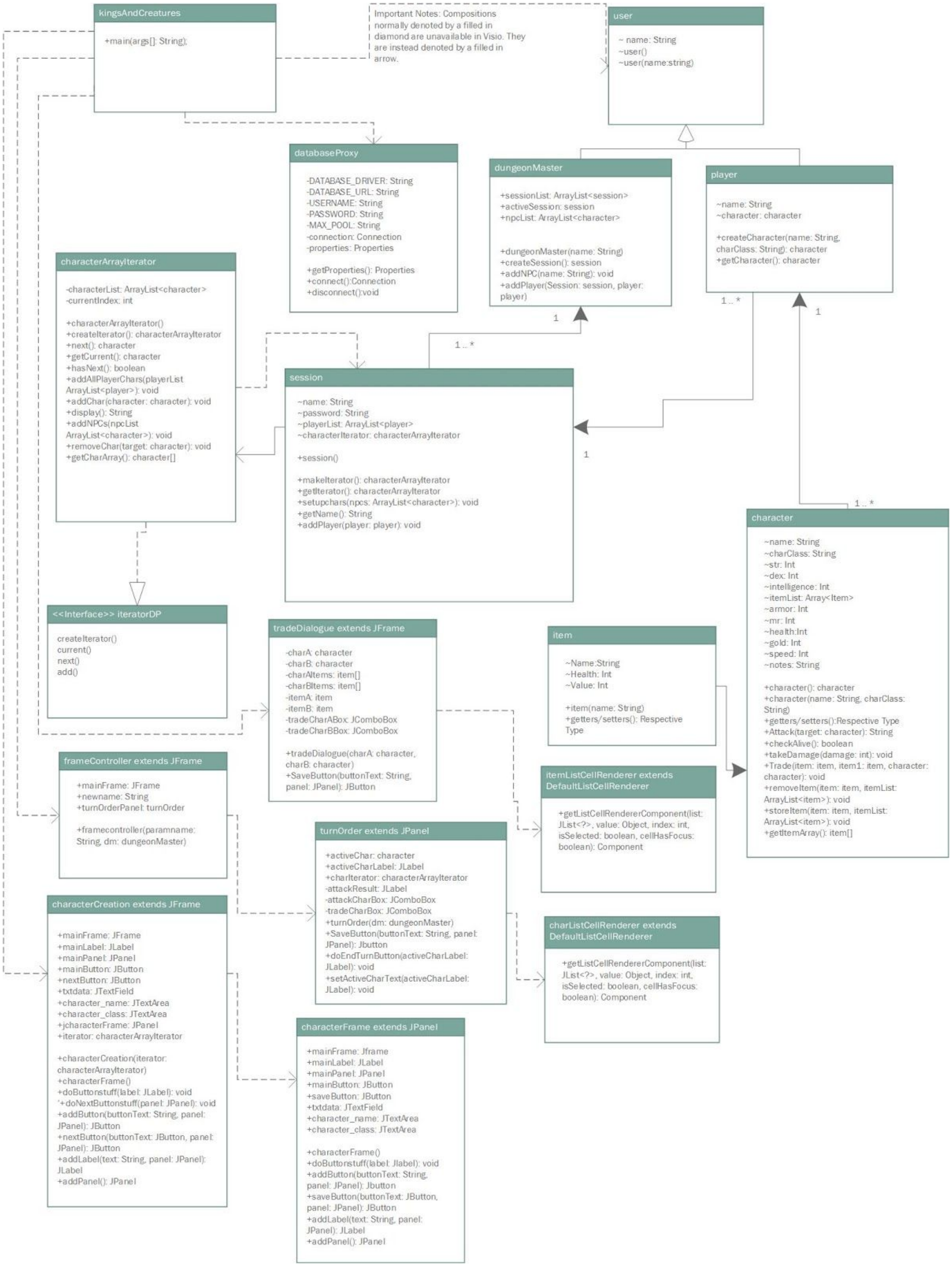
3a. Show your Part 2 class diagram and your final class diagram.

Part 2:



Important Notes: Compositions normally denoted by a filled in diamond are unavailable in Visio. They are instead denoted by a filled in arrow. Additionally all private variables will have getter and setter methods associated with them. Since private variables are clearly noted it is unnecessary to display them as part of our class diagram.

Final:


kingsAndCreatures
+main(args[]: String);
Important Notes: Compositions normally denoted by a filled in diamond are unavailable in Visio. They are instead denoted by a filled in arrow.
user
~ name: String
~user()
~user(name:string)
databaseProxy
-DATABASE_DRIVER: String
-DATABASE_URL: String
-USERNAME: String
-PASSWORD: String
-MAX_POOL: String
-connection: Connection
-properties: Properties
+getProperties(): Properties
+connect():Connection
+disconnect():void
dungeonMaster
+sessionList: ArrayList<session>
+activeSession: session
+npcList: ArrayList<character>
+dungeonMaster(name: String)
+createSession(): session
+addNPC(name: String): void
+addPlayer(Session: session, player: player)
player
~name: String
~character: character
+createCharacter(name: String, charClass: String): character
+getCharacter(): character
1 .. *
1
1
1 .. *
1
1 .. *
characterArrayIterator
-characterList: ArrayList<character>
-currentIndex: int
+characterArrayIterator()
+createIterator(): characterArrayIterator
+next(): character
+getCurrent(): character
+hasNext(): boolean
+addAllPlayerChars(playerList ArrayList<player>): void
+addChar(character: character): void
+display(): String
+addNPCs(npcList ArrayList<character>): void
+removeChar(target: character): void
+getCharArray(): character[]
session
~name: String
~password: String
~playerList: ArrayList<player>
~characterIterator: characterArrayIterator
+session()
+makeIterator(): characterArrayIterator
+getIterator(): characterArrayIterator
+setupchars(npcs: ArrayList<character>): void
+getName(): String
+addPlayer(player: player): void
1
character
~name: String
~charClass: String
~str: Int
~dex: Int
~intelligence: Int
~itemList: Array<Item>
~armor: Int
~mr: Int
~health:Int
~gold: Int
~speed: Int
~notes: String
+character(): character
+character(name: String, charClass: String)
+getters/setters():Respective Type
+Attack(target: character): String
+checkAlive(): boolean
+takeDamage(damage: int): void
+Trade(item: item, item1: item, character: character): void
+removeItem(item: item, itemList: ArrayList<item>): void
+storeItem(item: item, itemList: ArrayList<item>): void
+getItemArray(): item[]
<<Interface>> iteratorDP
createIterator()
current()
next()
add()
tradeDialogue extends JFrame
-charA: character
-charB: character
-charAItems: item[]
-charBItems: item[]
-itemA: item
-itemB: item
-tradeCharABox: JComboBox
-tradeCharBBox: JComboBox
+tradeDialogue(charA: character, charB: character)
+SaveButton(buttonText: String, panel: JPanel): JButton
item
~Name:String
~Health: Int
~Value: Int
+item(name: String)
+getters/setters(): Respective Type
itemListCellRenderer extends DefaultListCellRenderer
+getListCellRendererComponent(list: JList<?>, value: Object, index: int, isSelected: boolean, cellHasFocus: boolean): Component
frameController extends JFrame
+mainFrame: JFrame
+newname: String
+turnOrderPanel: turnOrder
+framecontroller(paramname: String, dm: dungeonMaster)
turnOrder extends JPanel
+activeChar: character
+activeCharLabel: JLabel
+charIterator: characterArrayIterator
-attackResult: JLabel
-attackCharBox: JComboBox
-tradeCharBox: JComboBox
+turnOrder(dm: dungeonMaster)
+SaveButton(buttonText: String, panel: JPanel): Jbutton
+doEndTurnButton(activeCharLabel: JLabel): void
+setActiveCharText(activeCharLabel: JLabel): void
charListCellRenderer extends DefaultListCellRenderer
+getListCellRendererComponent(list: JList<?>, value: Object, index: int, isSelected: boolean, cellHasFocus: boolean): Component
characterCreation extends JFrame
+mainFrame: JFrame
+mainLabel: JLabel
+mainPanel: JPanel
+mainButton: JButton
+nextButton: JButton
+txtdata: JTextField
+character_name: JTextArea
+character_class: JTextArea
+jcharacterFrame: JPanel
+iterator: characterArrayIterator
+characterCreation(iterator: characterArrayIterator)
+characterFrame()
+doButtonstuff(label: JLabel): void
'+doNextButtonstuff(panel: JPanel): void
+addButton(buttonText: String, panel: JPanel): JButton
+nextButton(buttonText: JButton, panel: JPanel): JButton
+addLabel(text: String, panel: JPanel): JLabel
+addPanel(): JPanel
characterFrame extends JPanel
+mainFrame: Jframe
+mainLabel: JLabel
+mainPanel: JPanel
+mainButton: JButton
+saveButton: JButton
+txtdata: JTextField
+character_name: JTextArea
+character_class: JTextArea
+characterFrame()
+doButtonstuff(label: Jlabel): void
+addButton(buttonText: String, panel: JPanel): Jbutton
+saveButton(buttonText: JButton, panel: JPanel): JButton
+addLabel(text: String, panel: JPanel): JLabel
+addPanel(): JPanel

3b. What changed? Why?

The primary change is that got rid of the UI controller class and implemented JFrame classes for the views. As a result, a majority of the new classes in the final class diagram are JFrame classes. The tiles class (and it's decorator and iterator) was not implemented due to time constraints. The room class was not implemented for the same reason. Most of the object oriented design is intact with what we designed in part 2 of the class diagram, however this is not entirely the case for the JFrame part of it. Since we didn't design it previously, alongside JFrame being new to us, this code does not entirely follow object oriented design.

3c. If it did not change much, then discuss how doing the design up front helped in the development.

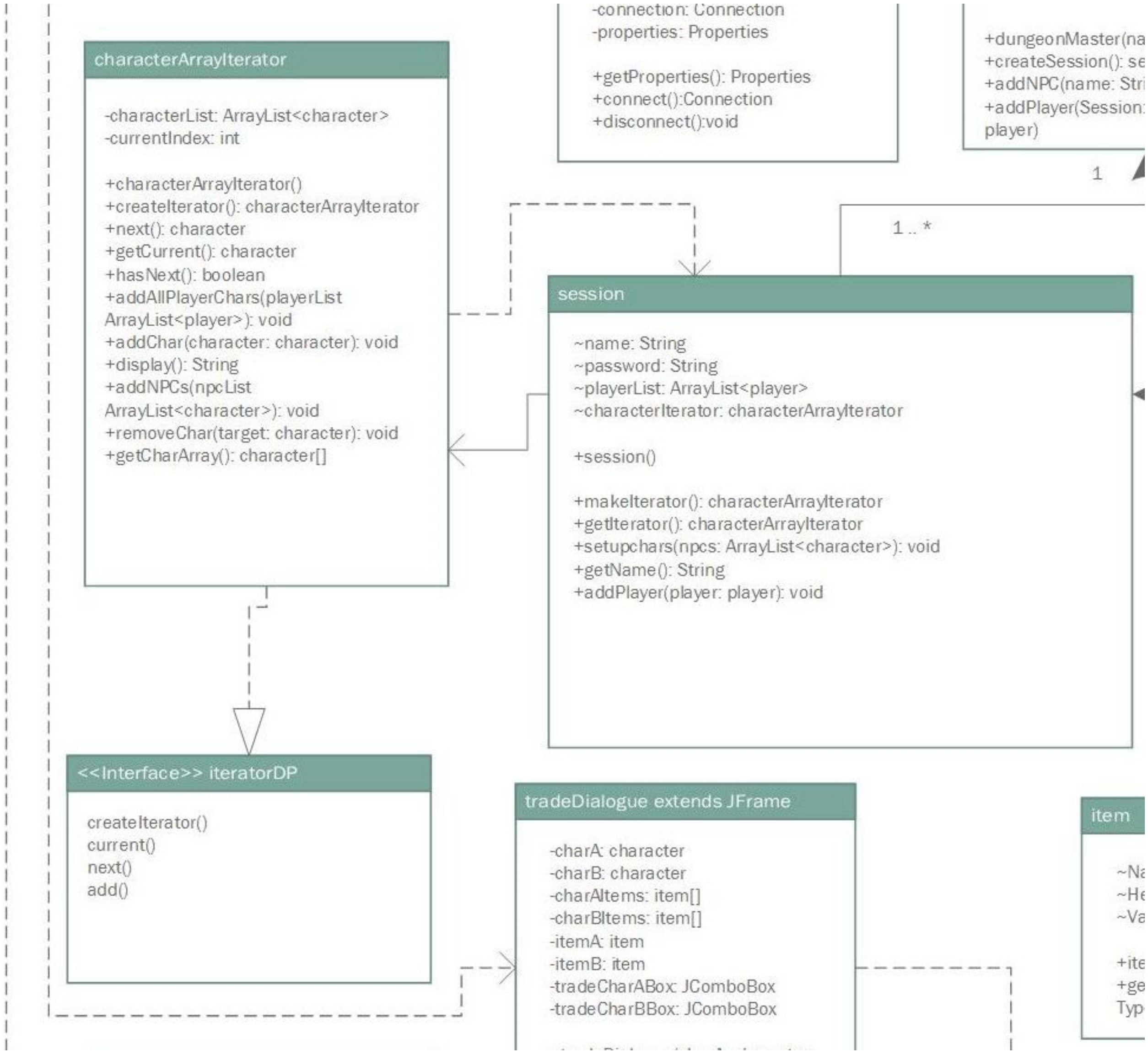
N/A

4a. Did you make use of any design patterns in the implementation of your final prototype?

We implemented an Iterator design pattern for turn order. The decorator design pattern for the tile class was not implemented.

4b. If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).

We used the Iterator design pattern to handle the turn order. This design pattern allowed us to add, remove and cycle through the characters to ensure that everyone had a turn.


-connection: Connection
-properties: Properties
+getProperties(): Properties
+connect():Connection
+disconnect():void
+dungeonMaster(na
+createSession(): se
+addNPC(name: Stri
+addPlayer(Session:
player)
1
1 .. *
characterArrayIterator
-characterList: ArrayList<character>
-currentIndex: int
+characterArrayIterator()
+createIterator(): characterArrayIterator
+next(): character
+getCurrent(): character
+hasNext(): boolean
+addAllPlayerChars(playerList
ArrayList<player>): void
+addChar(character: character): void
+display(): String
+addNPCs(npcList
ArrayList<character>): void
+removeChar(target: character): void
+getCharArray(): character[]
session
~name: String
~password: String
~playerList: ArrayList<player>
~characterIterator: characterArrayIterator
+session()
+makeIterator(): characterArrayIterator
+getIterator(): characterArrayIterator
+setupchars(npcs: ArrayList<character>): void
+getName(): String
+addPlayer(player: player): void
<<Interface>> iteratorDP
createIterator()
current()
next()
add()
tradeDialogue extends JFrame
-charA: character
-charB: character
-charAItems: item[]
-charBItems: item[]
-itemA: item
-itemB: item
-tradeCharABox: JComboBox
-tradeCharBBox: JComboBox
item
~Na
~He
~Va
+ite
+ge
Typ

4c. If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

NA

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?

We have learned that the creation of a system requires a lot of planning. We have also learned that there are many different ways to create a system, all of which can work, but some do not work together very well. Originally we didn’t consider using JFrame, so our class diagram didn’t contain any of the necessary functions and classes for it’s implementation. As a result, our JFrame code became rather cluttered. Whereas the classes and methods we designed in the class diagram were simple and easy to implement.