1. **Name:** Katherine Best
2. **Description:** A very basic poker game that players will be able to play locally. It is loosely based on Texas Hold'em poker but does not follow all the rules and conventions of traditional Texas Hold'em for simplicity.
3. **Features Implemented:**

Requirement ID	Description
UR 1	Players must be able to see the current value of the pot.
UR 2	Players must be able to see the current bet.
UR 3	Players must be able to see the current round.
UR 4	Players must be able to fold on their betting turn.
UR 5	Players must be able to call on their betting turn.
UR 6	Players must be able to raise on their betting turn.
UR 7	Players must be able to specify the amount of the raise when they choose to raise on their betting round.
UR 8	Players must be able to choose to save the game after each betting round.[1]
UR 9	Players must be able to specify the hand they want to play after the last betting round.
FR 1	After a player raises, all previous players must have the chance to bet again.
FR 2	Values of hands must be compared.
FR 3	The player with the highest value hand must be declared the winner.

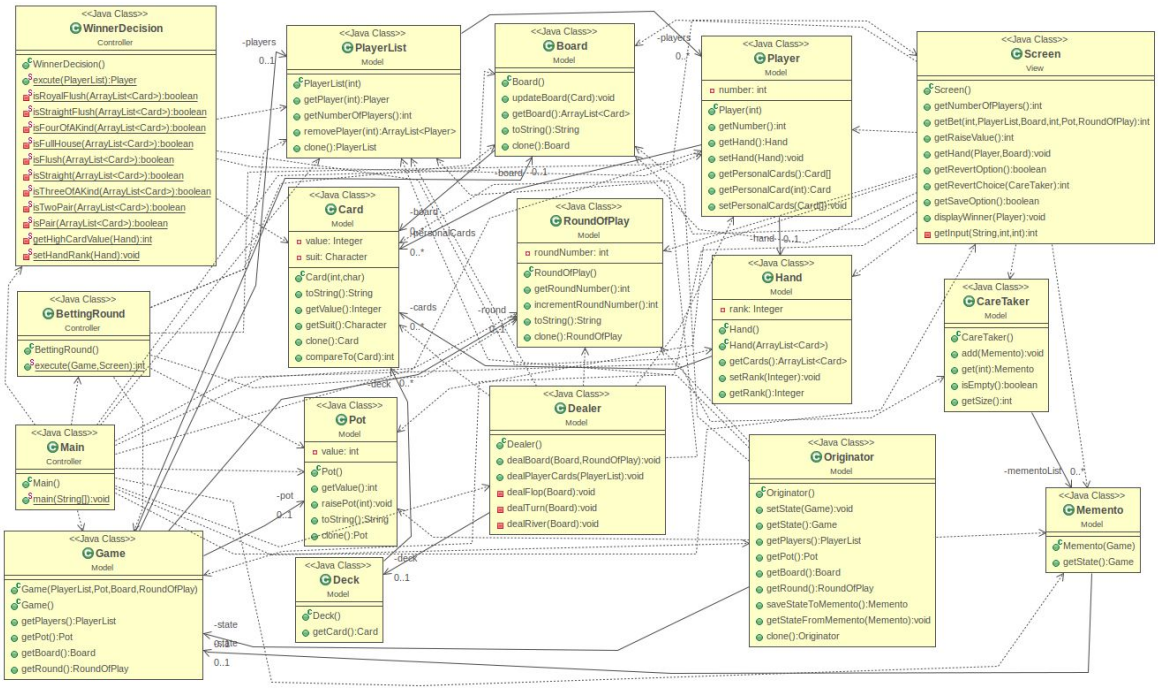
4. **Features Not Implemented:** All features listed in the initial specification of the project (submitted as part of Part 2) were implemented in this project. The following features are examples of features that could be implemented in future versions.

Requirement ID	Description
UR 10	Players must be able to specify how much money they are starting the game with.

[1] Note that this requirement was changed from the initial requirements list. It now says that players should be able to save after each betting round instead of after each deal.

UR 11	Players must be able to see how much money they have left.
FR 4	The highest hand that can be made from the players cards and the board cards must be used as the players hand. (This requirement would replace UR 9.)

5. **Final Class Diagram:**

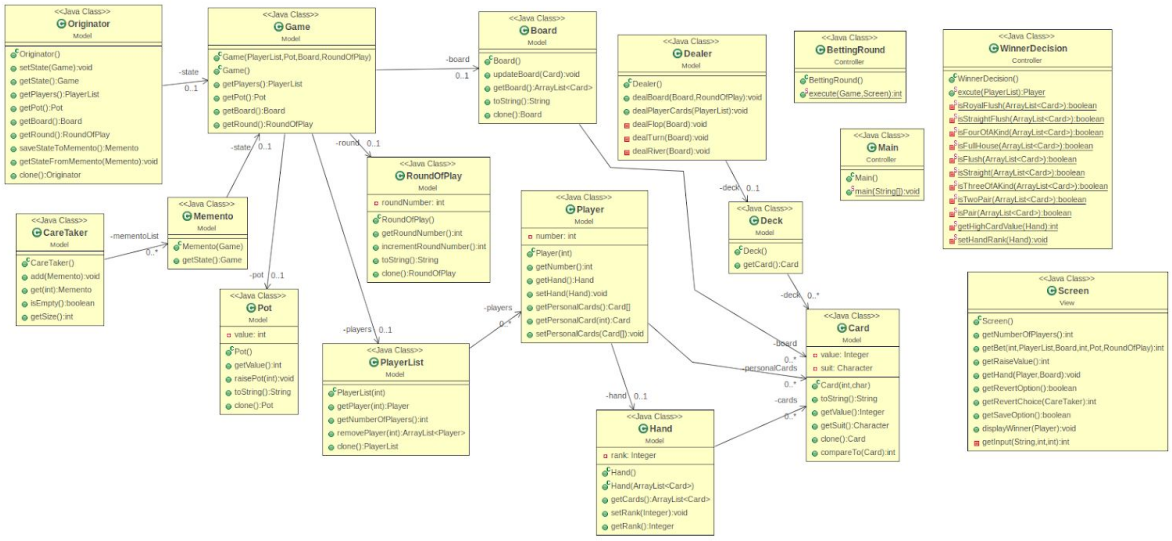


There are more classes in the model and controller in the final version than there were in the initial version. The model has more classes partly because the initial diagram did not contain the classes for the design pattern. The inclusion of the design pattern necessitated the addition of the Originator, Memento, and CareTaker classes. Also the Game class was added to hold the state of the game and make the implementations of the other three classes cleaner. Other than the classes that were added for the design pattern, the PlayerList and Hand classes were also added. The PlayerList class was added to more neatly contain the list of players currently in the game and to provide a simpler way to perform common operations on the list of players. The Hand class was added so that the players hands could be ranked against one another and as a more convenient way to handle the players' hands.

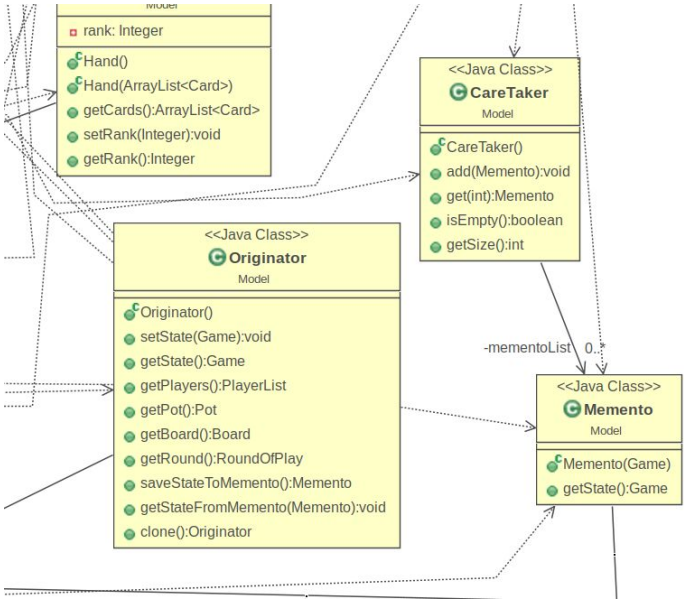
Notes:

- The view is shown in this diagram but it was not shown in the first class diagram.
- Private and public attributes are denoted with green and red markers respectively. So, the green markers should be "+" symbols and the red markers should be "-" symbols.

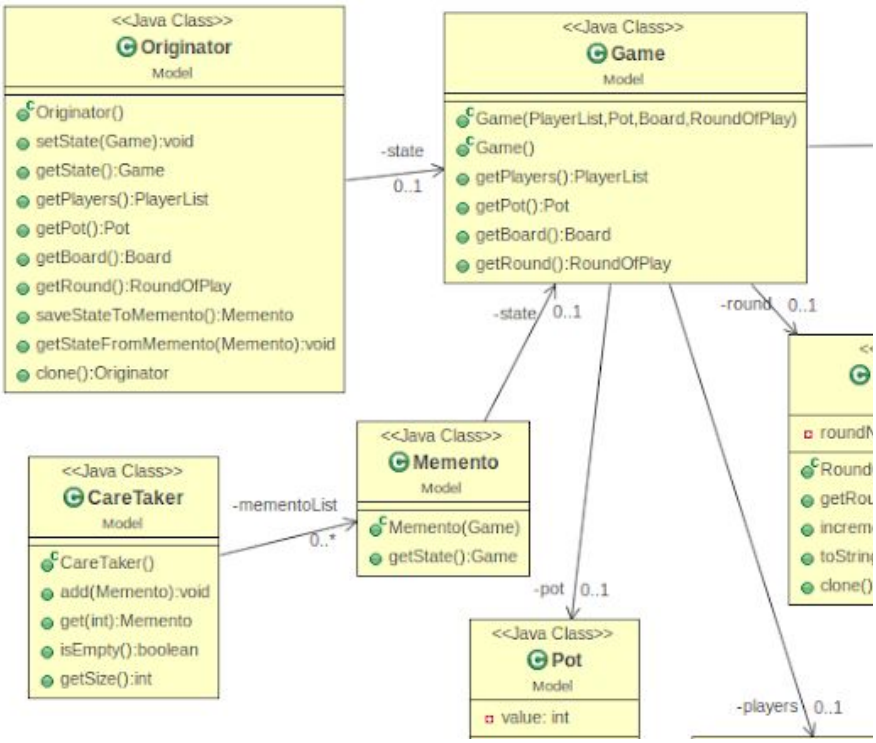
- To conserve space, if there is a composite class that has an attribute that is an instance of another class shown in the class diagram, the attribute label is not shown in the box of the class but is instead next to the arrow connecting the two classes. If the attribute is by an arrow the visibility is marked with the standard "+" and "-" symbols.
- A simplified class diagram that does not show dependencies is included below
- PDFs of both the full and simplified class diagrams are included at the end of this document



6. **Design Pattern:** *Memento*
   - Classes from my class diagram that implement the pattern:



Note that the Memento object has an arrow to the Game object that we do not see here because the Game object is on the other side of the class diagram. We can see this in the simplified class diagram that follows.



- Class diagram for the memento pattern:

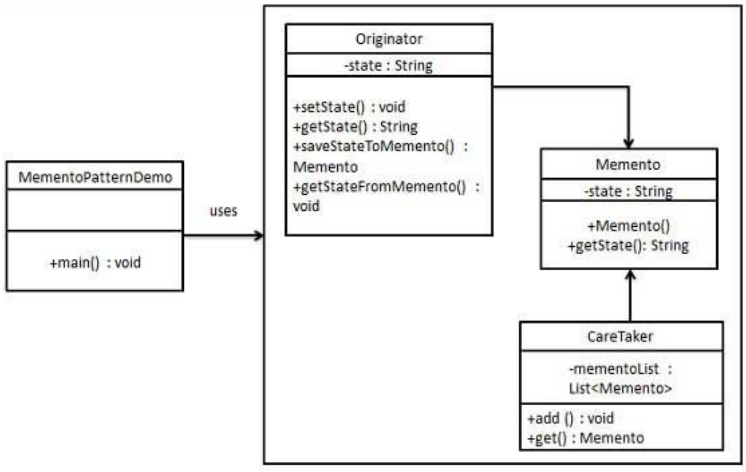


Image from Tutorials Point : https://www.tutorialspoint.com/design_pattern/memento_pattern.htm

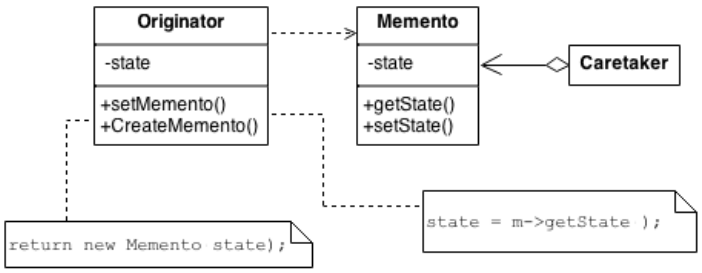
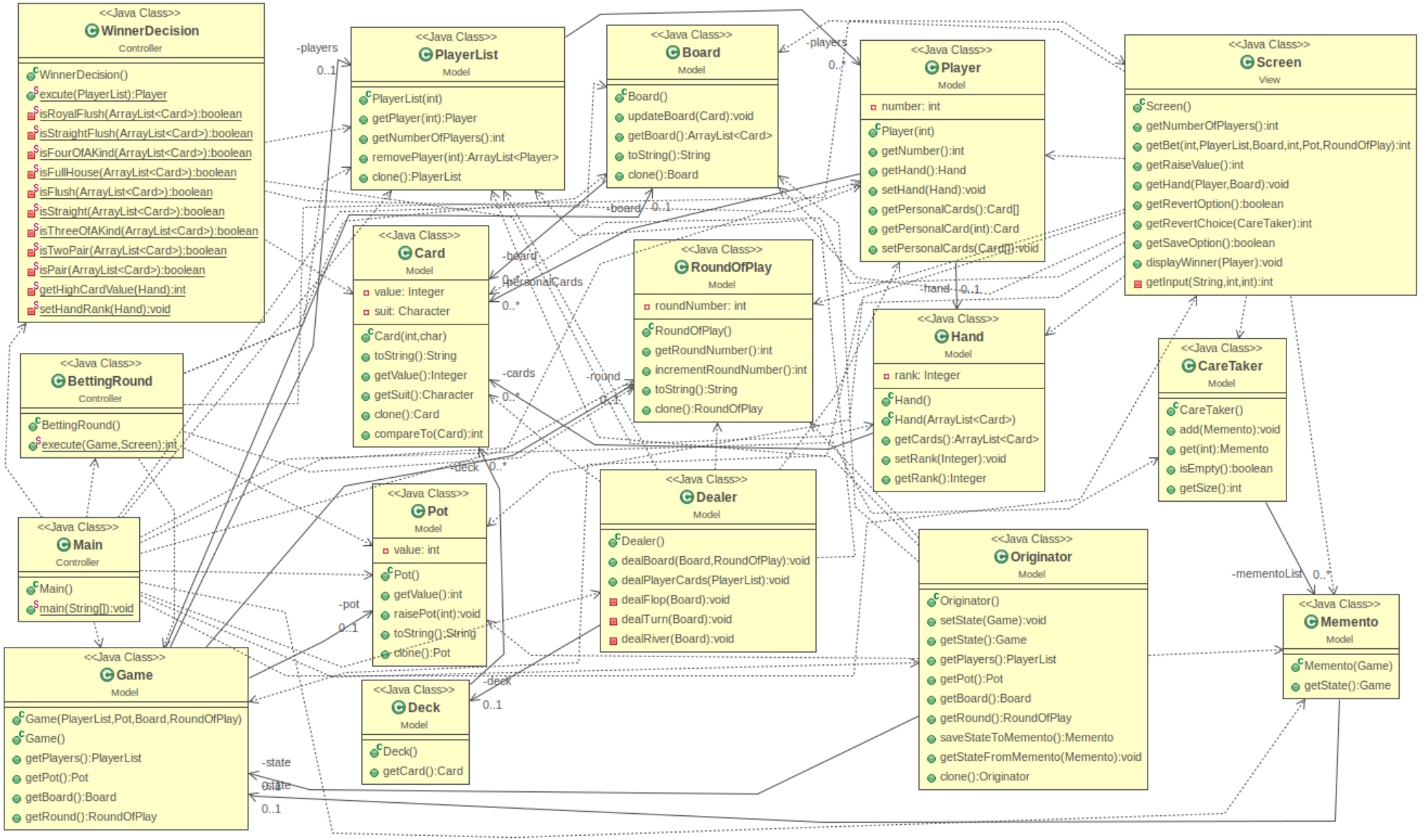
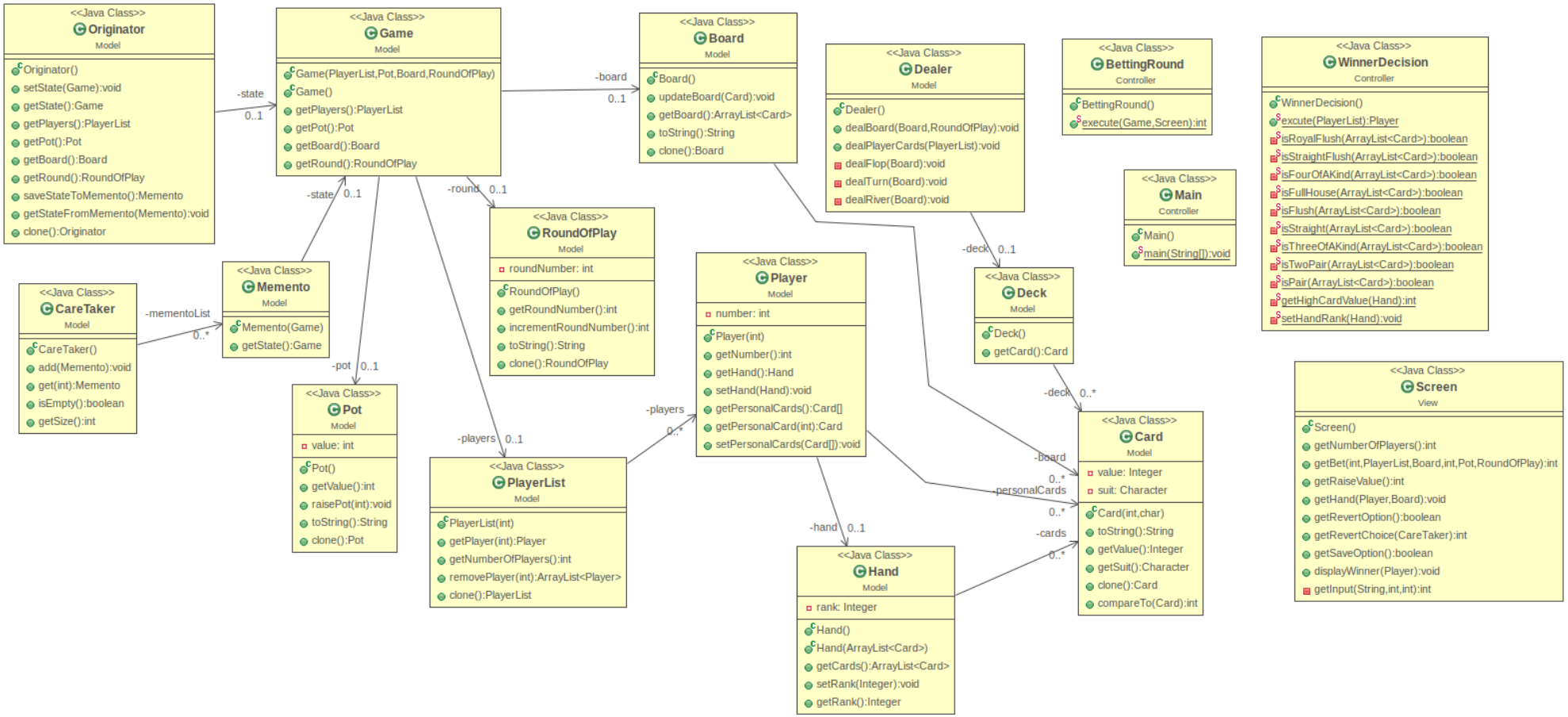


Image from Source Making :
https://sourcemaking.com/design_patterns/memento

- How and why I used the memento design pattern:
  I used the memento design pattern because I wanted the users to be able to save their game and go back to an earlier point in the game and the memento design pattern is an effective way to implement this. There are three main classes that are used in the memento design pattern; Originator, Memento, and CareTaker. The Memento class is an object that holds a game state. (In my case, the state was a Game object since the Game class holds all the information necessary to determine the state of the game.) The CareTaker keeps track of the saved game states. It holds the Mementos for the game states that have been saved. The Originator has the state of the current game and can save itself to a Memento.

7. **What Have You Learned:**
   I have learned a great deal about the process of analysis and design now that I have gone through the steps of creating, designing, and implementing a system. In particular, I have learned a lot about designing a system. Prior to this experience I have never spent the time to think out how the system I was building should work and what the best way to implement it would be. I think planning this out beforehand was a great experience and it will be a valuable tool in the future. I think I have learned how important this initial planning phase can be because it makes the implementation of the system much easier and it makes the overall quality of the system much better.


<<Java Class>>
WinnerDecision
Controller
WinnerDecision()
excute(PlayerList):Player
isRoyalFlush(ArrayList<Card>):boolean
isStraightFlush(ArrayList<Card>):boolean
isFourOfAKind(ArrayList<Card>):boolean
isFullHouse(ArrayList<Card>):boolean
isFlush(ArrayList<Card>):boolean
isStraight(ArrayList<Card>):boolean
isThreeOfAKind(ArrayList<Card>):boolean
isTwoPair(ArrayList<Card>):boolean
isPair(ArrayList<Card>):boolean
getHighCardValue(Hand):int
setHandRank(Hand):void
<<Java Class>>
PlayerList
Model
PlayerList(int)
getPlayer(int):Player
getNumberOfPlayers():int
removePlayer(int):ArrayList<Player>
clone():PlayerList
<<Java Class>>
Board
Model
Board()
updateBoard(Card):void
getBoard():ArrayList<Card>
toString():String
clone():Board
<<Java Class>>
Player
Model
number: int
Player(int)
getNumber():int
getHand():Hand
setHand(Hand):void
getPersonalCards():Card[]
getPersonalCard(int):Card
setPersonalCards(Card[]):void
<<Java Class>>
Screen
View
Screen()
getNumberOfPlayers():int
getBet(int,PlayerList,Board,int,Pot,RoundOfPlay):int
getRaiseValue():int
getHand(Player,Board):void
getRevertOption():boolean
getRevertChoice(CareTaker):int
getSaveOption():boolean
displayWinner(Player):void
getInput(String,int,int):int
<<Java Class>>
Card
Model
value: Integer
suit: Character
Card(int,char)
toString():String
getValue():Integer
getSuit():Character
clone():Card
compareTo(Card):int
<<Java Class>>
RoundOfPlay
Model
roundNumber: int
RoundOfPlay()
getRoundNumber():int
incrementRoundNumber():int
toString():String
clone():RoundOfPlay
<<Java Class>>
Hand
Model
rank: Integer
Hand()
Hand(ArrayList<Card>)
getCards():ArrayList<Card>
setRank(Integer):void
getRank():Integer
<<Java Class>>
CareTaker
Model
CareTaker()
add(Memento):void
get(int):Memento
isEmpty():boolean
getSize():int
<<Java Class>>
BettingRound
Controller
BettingRound()
execute(Game,Screen):int
<<Java Class>>
Main
Controller
Main()
main(String[]):void
<<Java Class>>
Pot
Model
value: int
Pot()
getValue():int
raisePot(int):void
toString():String
clone():Pot
<<Java Class>>
Dealer
Model
Dealer()
dealBoard(Board,RoundOfPlay):void
dealPlayerCards(PlayerList):void
dealFlop(Board):void
dealTurn(Board):void
dealRiver(Board):void
<<Java Class>>
Originator
Model
Originator()
setState(Game):void
getState():Game
getPlayers():PlayerList
getPot():Pot
getBoard():Board
getRound():RoundOfPlay
saveStateToMemento():Memento
getStateFromMemento(Memento):void
clone():Originator
<<Java Class>>
Memento
Model
Memento(Game)
getState():Game
<<Java Class>>
Game
Model
Game(PlayerList,Pot,Board,RoundOfPlay)
Game()
getPlayers():PlayerList
getPot():Pot
getBoard():Board
getRound():RoundOfPlay
<<Java Class>>
Deck
Model
Deck()
getCard():Card
-players
0..1
-players
0..*
-board
0..1
-hand
0..1
-personalCards
0..*
-cards
0..*
-round
0..1
-deck
0..*
-pot
0..1
-deck
0..1
-mementoList
0..*
-state
0..1
-state
0..1


<<Java Class>>
Originator
Model
Originator()
setState(Game):void
getState():Game
getPlayers():PlayerList
getPot():Pot
getBoard():Board
getRound():RoundOfPlay
saveStateToMemento():Memento
getStateFromMemento(Memento):void
clone():Originator
-state
0..1
<<Java Class>>
Game
Model
Game(PlayerList,Pot,Board,RoundOfPlay)
Game()
getPlayers():PlayerList
getPot():Pot
getBoard():Board
getRound():RoundOfPlay
-board
0..1
<<Java Class>>
Board
Model
Board()
updateBoard(Card):void
getBoard():ArrayList<Card>
toString():String
clone():Board
<<Java Class>>
Dealer
Model
Dealer()
dealBoard(Board,RoundOfPlay):void
dealPlayerCards(PlayerList):void
dealFlop(Board):void
dealTurn(Board):void
dealRiver(Board):void
<<Java Class>>
BettingRound
Controller
BettingRound()
execute(Game,Screen):int
<<Java Class>>
WinnerDecision
Controller
WinnerDecision()
excute(PlayerList):Player
isRoyalFlush(ArrayList<Card>):boolean
isStraightFlush(ArrayList<Card>):boolean
isFourOfAKind(ArrayList<Card>):boolean
isFullHouse(ArrayList<Card>):boolean
isFlush(ArrayList<Card>):boolean
isStraight(ArrayList<Card>):boolean
isThreeOfAKind(ArrayList<Card>):boolean
isTwoPair(ArrayList<Card>):boolean
isPair(ArrayList<Card>):boolean
getHighCardValue(Hand):int
setHandRank(Hand):void
<<Java Class>>
Main
Controller
Main()
main(String[]):void
-state
0..1
-round
0..1
<<Java Class>>
RoundOfPlay
Model
roundNumber: int
RoundOfPlay()
getRoundNumber():int
incrementRoundNumber():int
toString():String
clone():RoundOfPlay
-deck
0..1
<<Java Class>>
Deck
Model
Deck()
getCard():Card
<<Java Class>>
CareTaker
Model
CareTaker()
add(Memento):void
get(int):Memento
isEmpty():boolean
getSize():int
-mementoList
0..*
<<Java Class>>
Memento
Model
Memento(Game)
getState():Game
-pot
0..1
<<Java Class>>
Pot
Model
value: int
Pot()
getValue():int
raisePot(int):void
toString():String
clone():Pot
-players
0..1
<<Java Class>>
PlayerList
Model
PlayerList(int)
getPlayer(int):Player
getNumberOfPlayers():int
removePlayer(int):ArrayList<Player>
clone():PlayerList
-players
0..*
<<Java Class>>
Player
Model
number: int
Player(int)
getNumber():int
getHand():Hand
setHand(Hand):void
getPersonalCards():Card[]
getPersonalCard(int):Card
setPersonalCards(Card[]):void
-hand
0..1
<<Java Class>>
Hand
Model
rank: Integer
Hand()
Hand(ArrayList<Card>)
getCards():ArrayList<Card>
setRank(Integer):void
getRank():Integer
-deck
0..*
-board
0..*
-personalCards
0..*
-cards
0..*
<<Java Class>>
Card
Model
value: Integer
suit: Character
Card(int,char)
toString():String
getValue():Integer
getSuit():Character
clone():Card
compareTo(Card):int
<<Java Class>>
Screen
View
Screen()
getNumberOfPlayers():int
getBet(int,PlayerList,Board,int,Pot,RoundOfPlay):int
getRaiseValue():int
getHand(Player,Board):void
getRevertOption():boolean
getRevertChoice(CareTaker):int
getSaveOption():boolean
displayWinner(Player):void
getInput(String,int,int):int