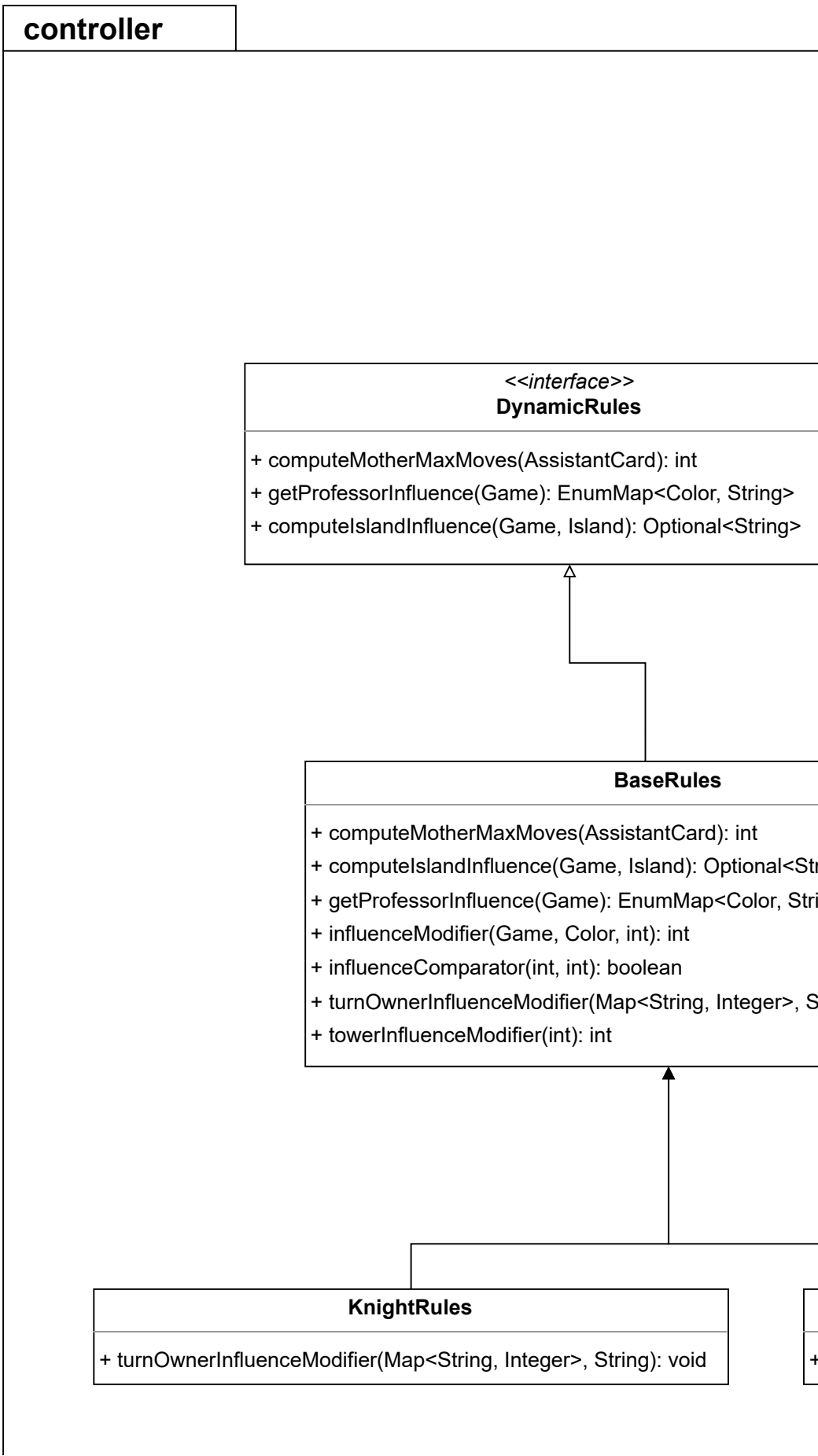
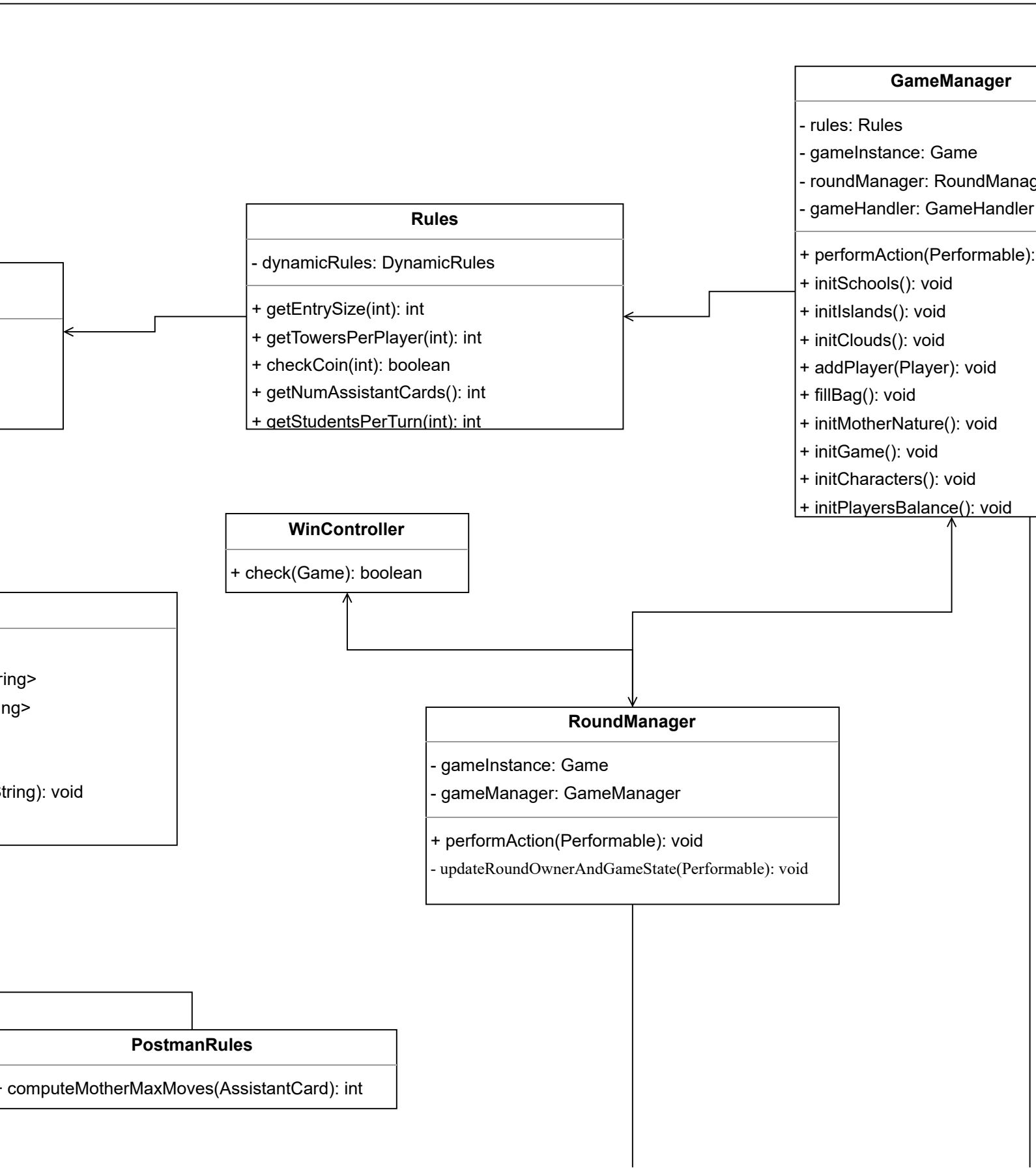
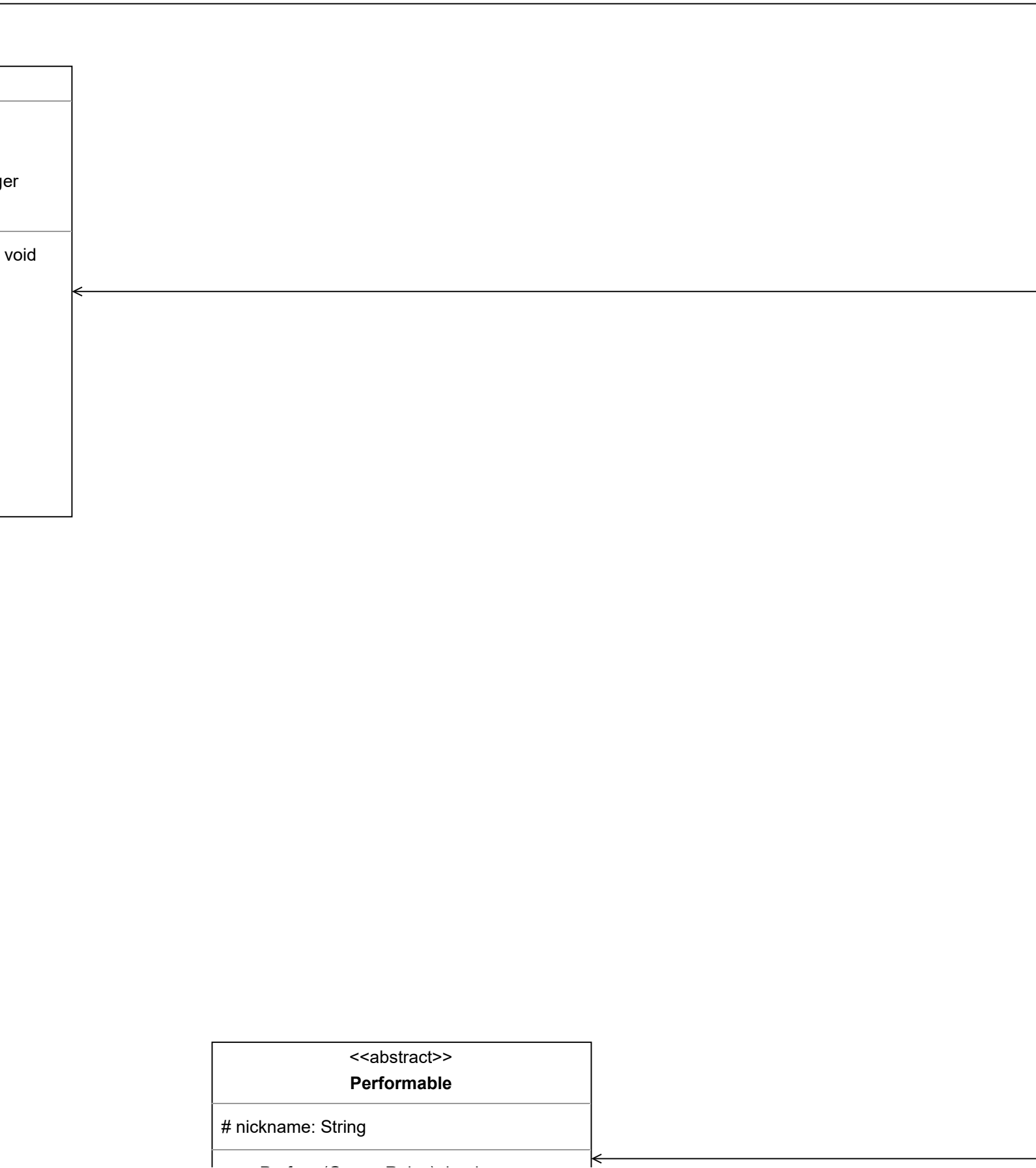
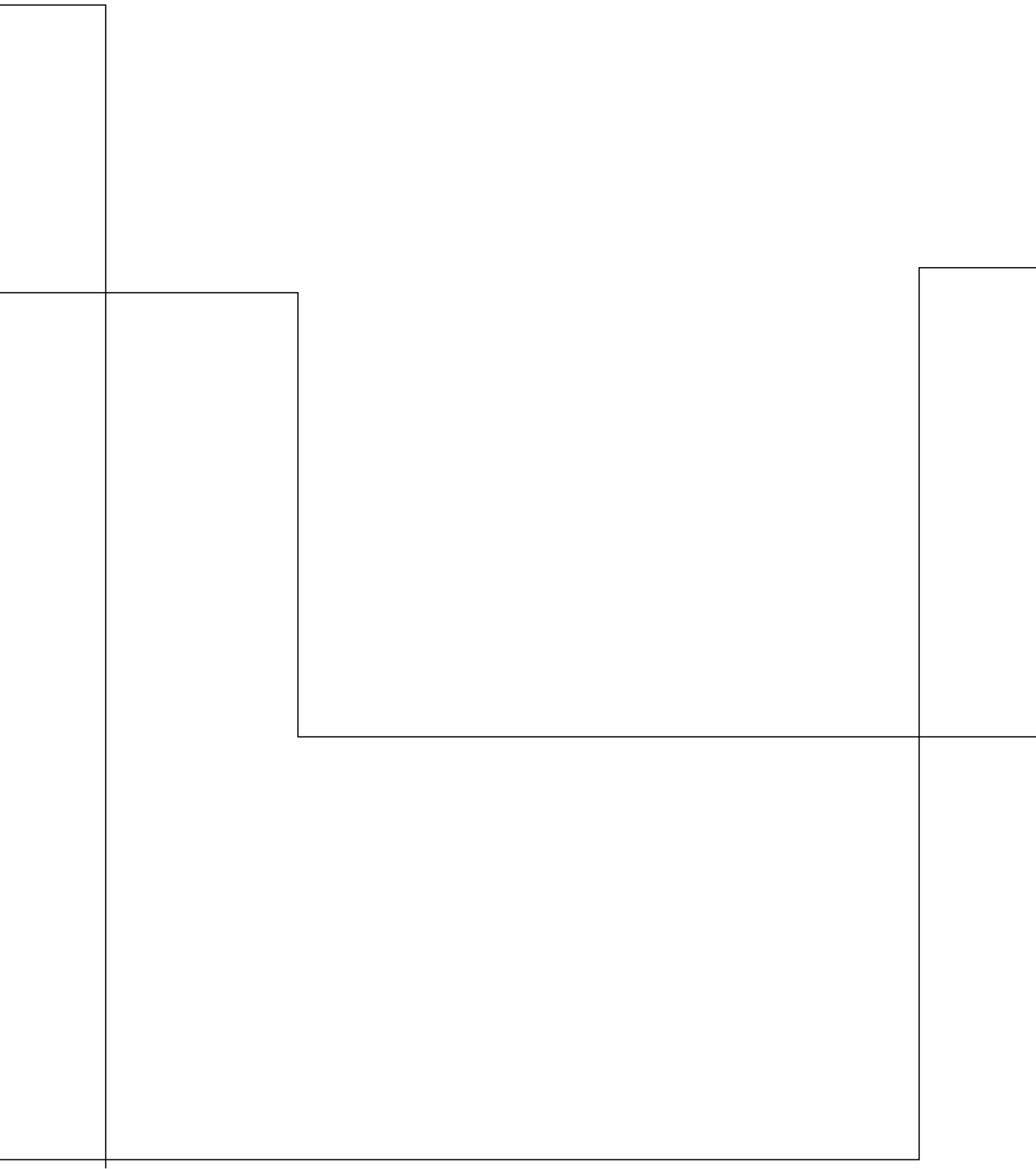
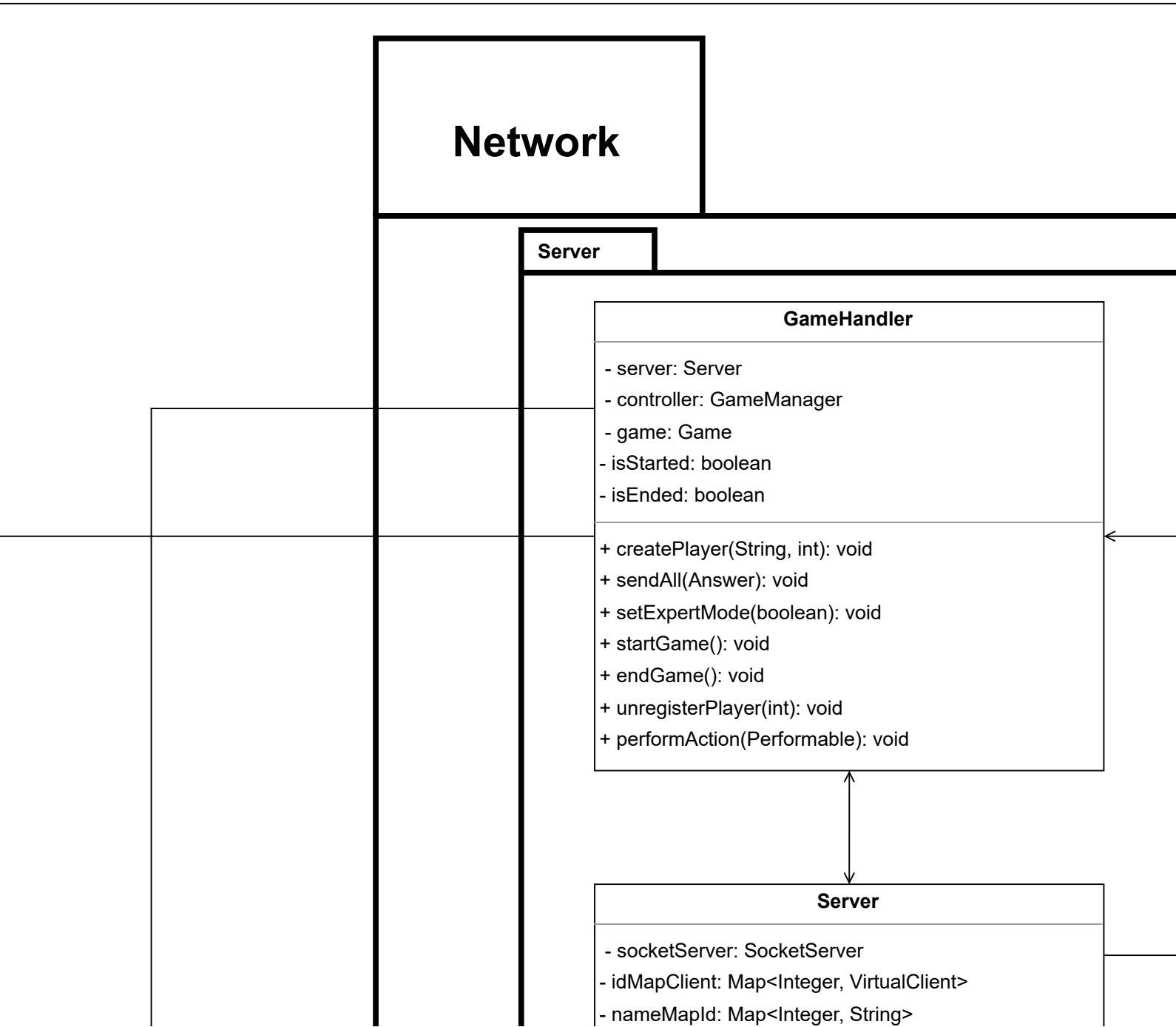
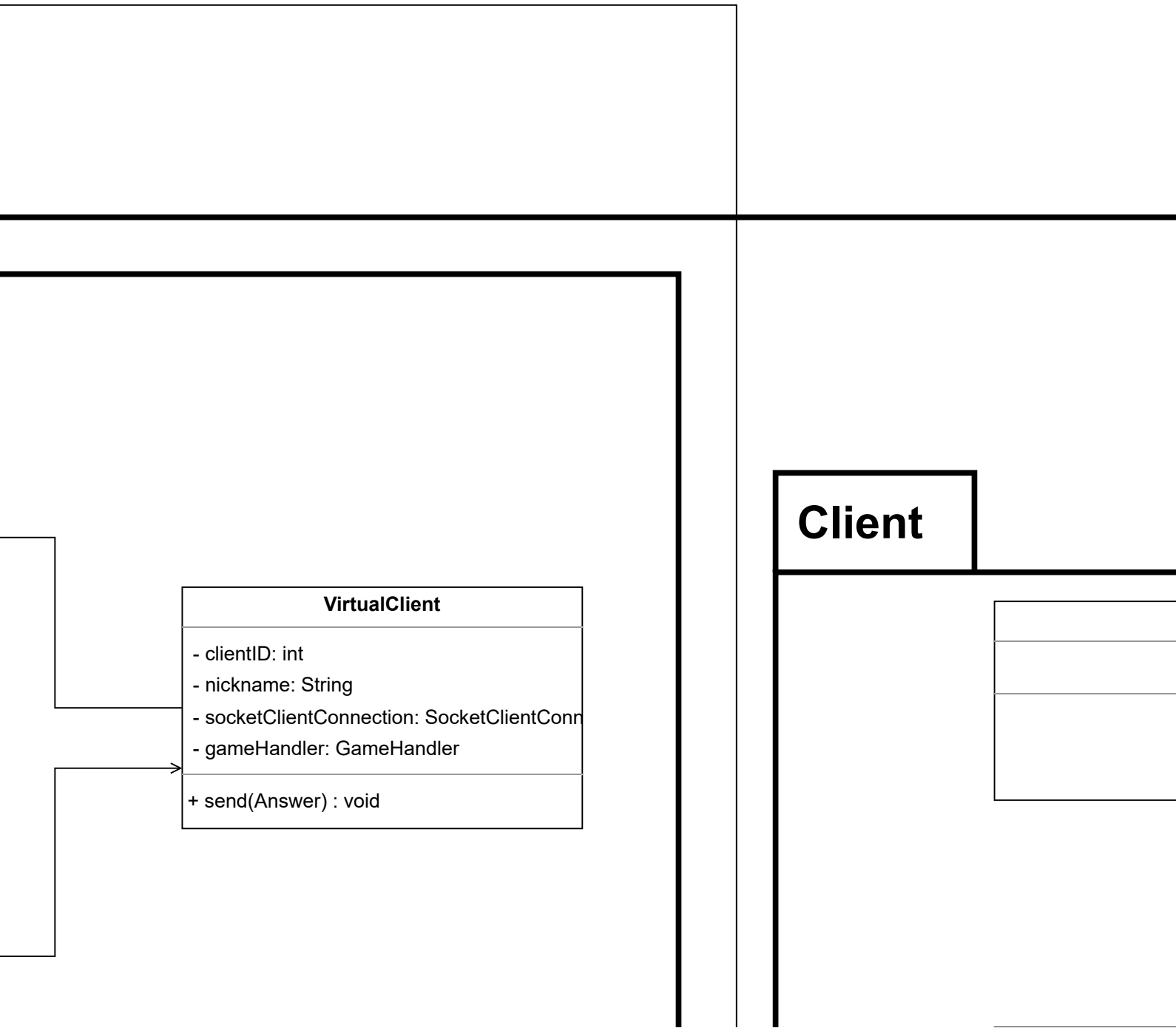

controller
<<interface>>
DynamicRules
+ computeMotherMaxMoves(AssistantCard): int
+ getProfessorInfluence(Game): EnumMap<Color, String>
+ computeIslandInfluence(Game, Island): Optional<String>
BaseRules
+ computeMotherMaxMoves(AssistantCard): int
+ computeIslandInfluence(Game, Island): Optional<Str
+ getProfessorInfluence(Game): EnumMap<Color, Stri
+ influenceModifier(Game, Color, int): int
+ influenceComparator(int, int): boolean
+ turnOwnerInfluenceModifier(Map<String, Integer>, S
+ towerInfluenceModifier(int): int
KnightRules
+ turnOwnerInfluenceModifier(Map<String, Integer>, String): void

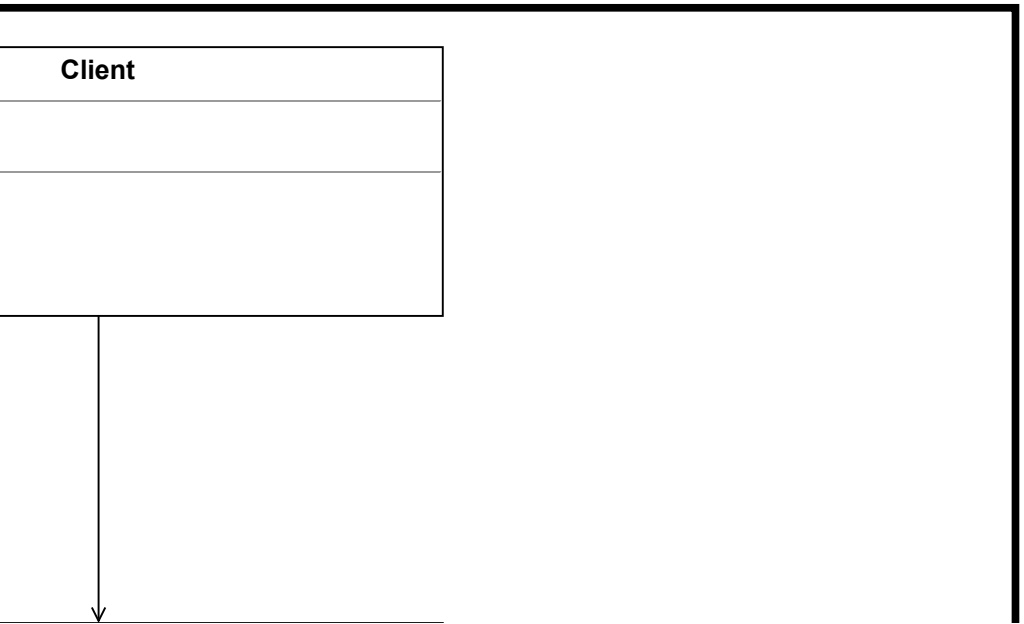

GameManager
- rules: Rules
- gameInstance: Game
- roundManager: RoundManag
- gameHandler: GameHandler
+ performAction(Performable):
+ initSchools(): void
+ initIslands(): void
+ initClouds(): void
+ addPlayer(Player): void
+ fillBag(): void
+ initMotherNature(): void
+ initGame(): void
+ initCharacters(): void
+ initPlayersBalance(): void
Rules
- dynamicRules: DynamicRules
+ getEntrySize(int): int
+ getTowersPerPlayer(int): int
+ checkCoin(int): boolean
+ getNumAssistantCards(): int
+ getStudentsPerTurn(int): int
WinController
+ check(Game): boolean
ring>
ng>
tring): void
RoundManager
- gameInstance: Game
- gameManager: GameManager
+ performAction(Performable): void
- updateRoundOwnerAndGameState(Performable): void
PostmanRules
- computeMotherMaxMoves(AssistantCard): int

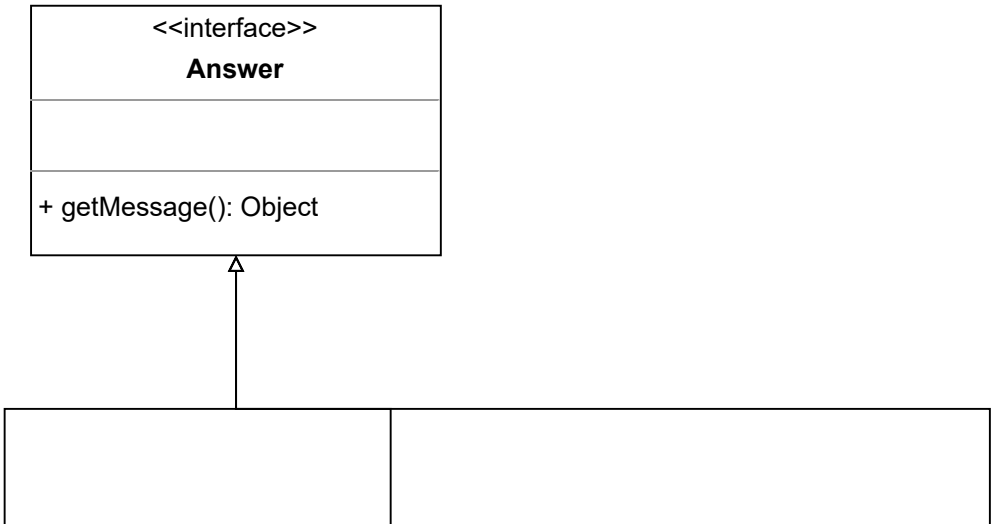

ger
void
<<abstract>>
Performable
# nickname: String

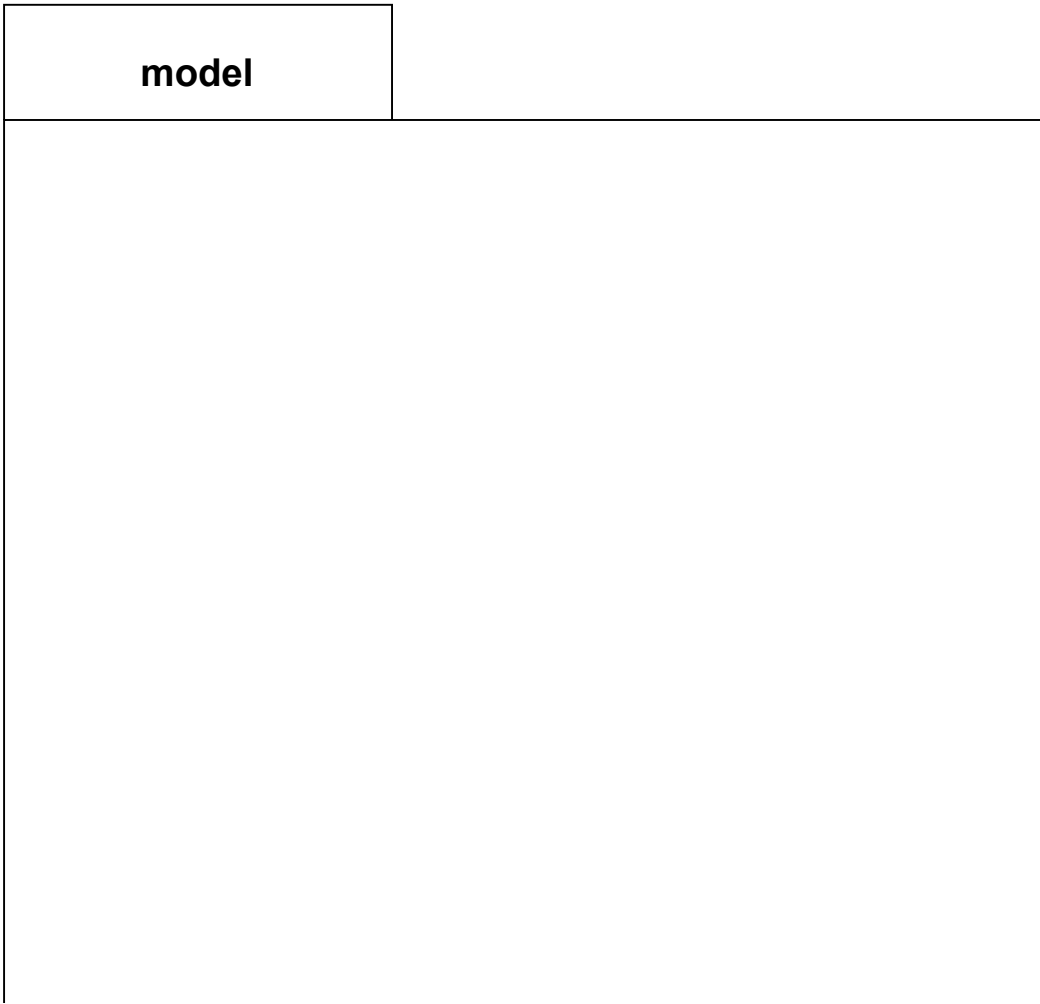


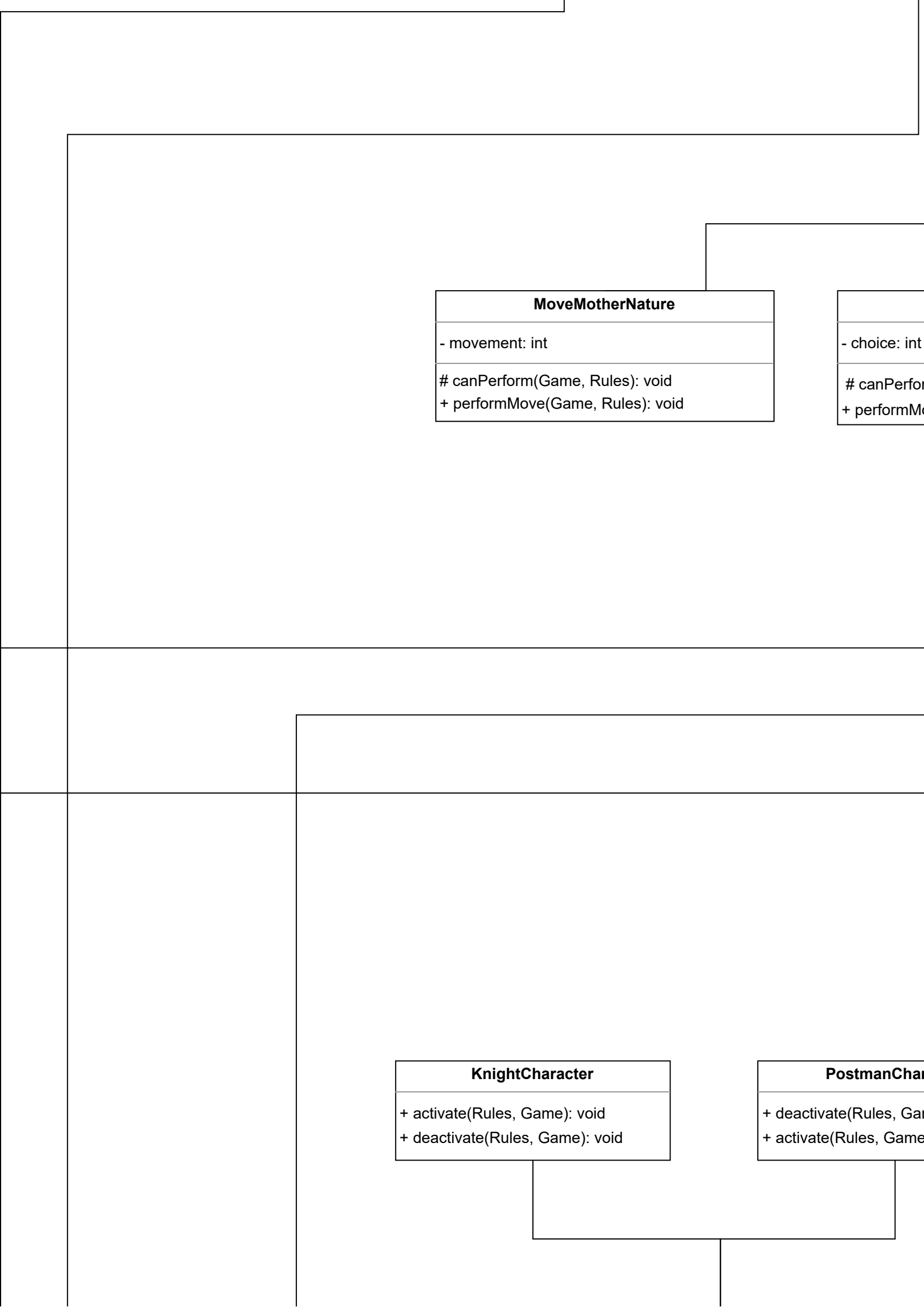

Network
Server
GameHandler
- server: Server
- controller: GameManager
- game: Game
- isStarted: boolean
- isEnded: boolean
+ createPlayer(String, int): void
+ sendAll(Answer): void
+ setExpertMode(boolean): void
+ startGame(): void
+ endGame(): void
+ unregisterPlayer(int): void
+ performAction(Performable): void
Server
- socketServer: SocketServer
- idMapClient: Map<Integer, VirtualClient>
- nameMapId: Map<Integer, String>

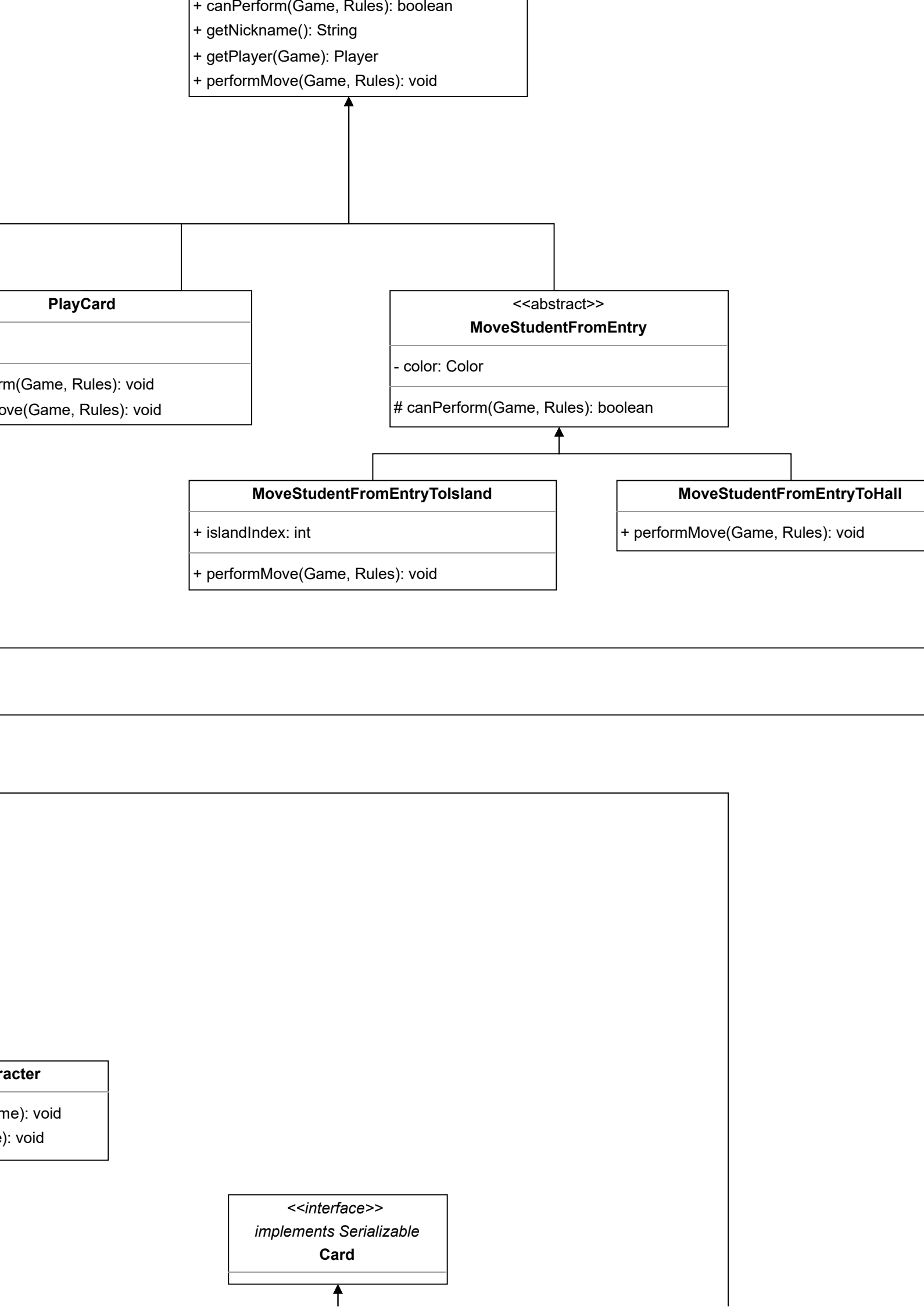

Client
VirtualClient
- clientID: int
- nickname: String
- socketClientConnection: SocketClientConn
- gameHandler: GameHandler
+ send(Answer) : void

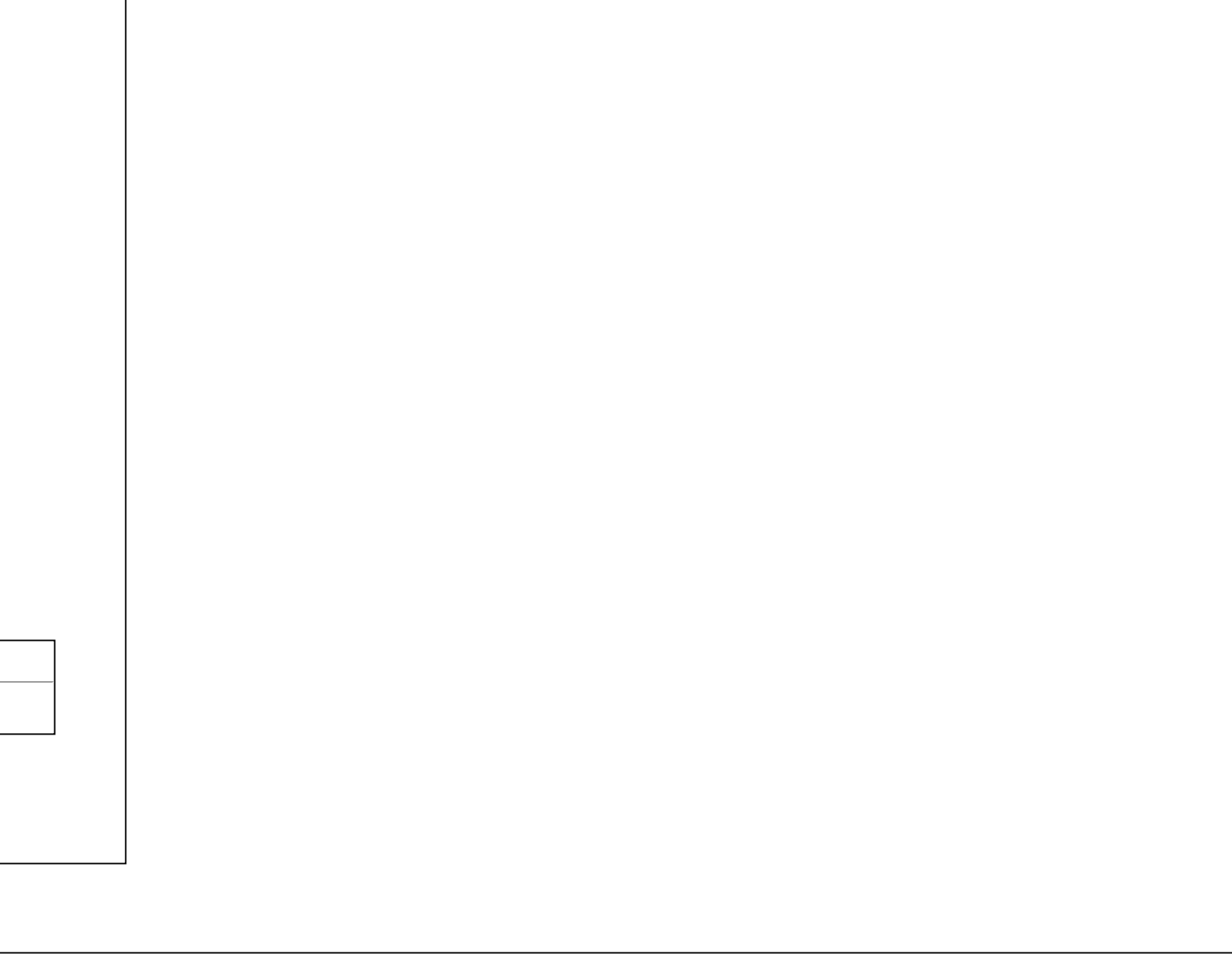

Client

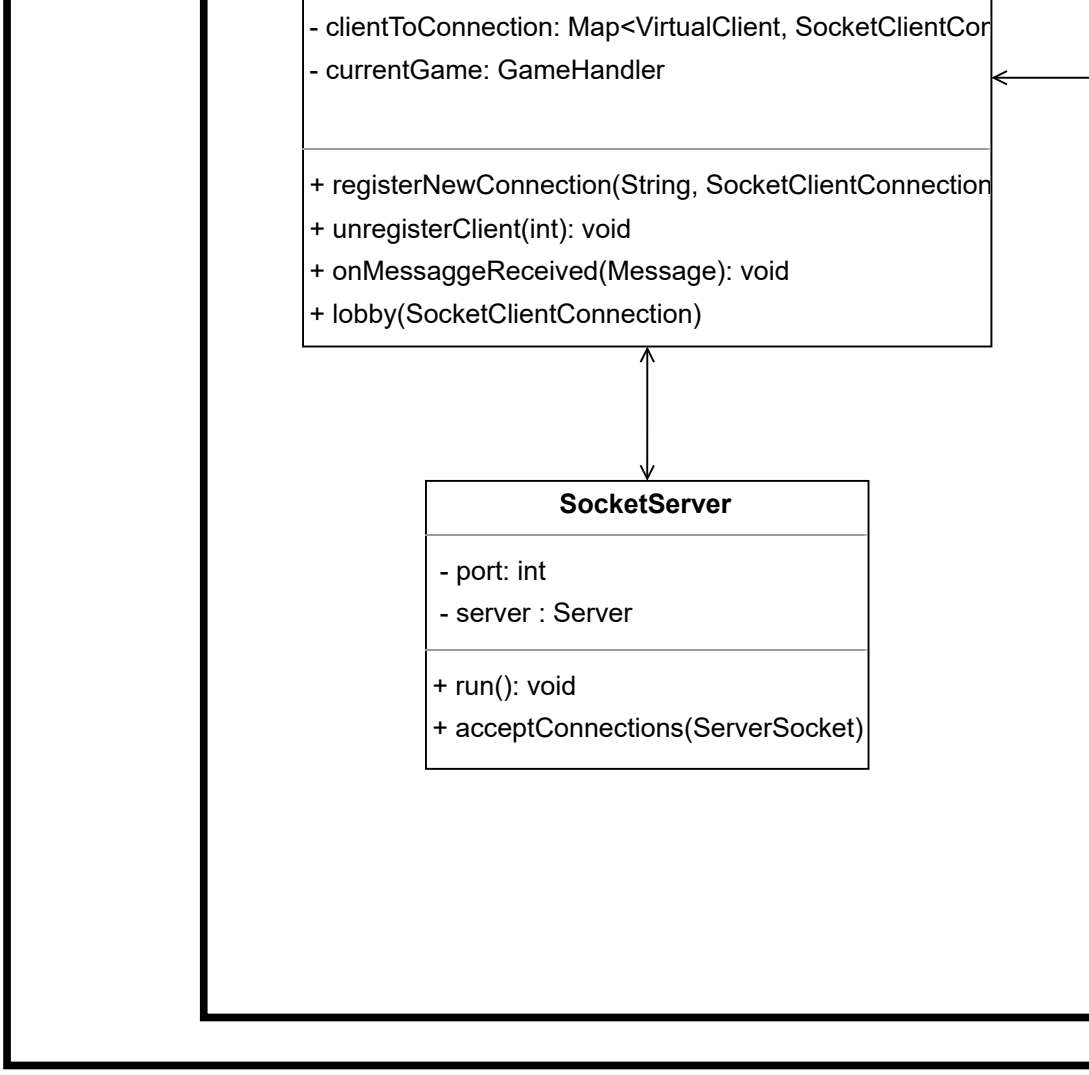

<<interface>>
Answer
+ getMessage(): Object

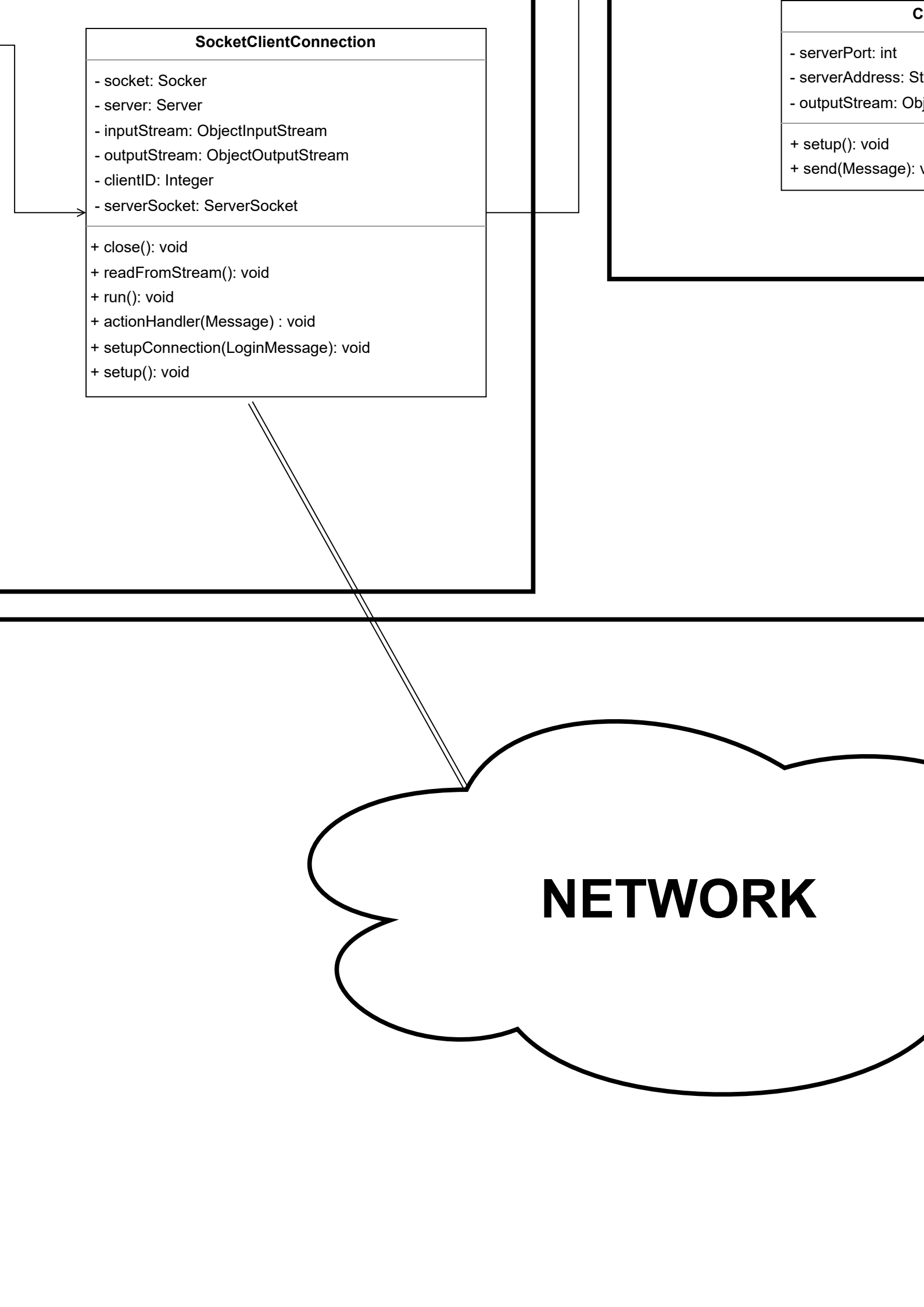

model


MoveMotherNature
- movement: int
# canPerform(Game, Rules): void
+ performMove(Game, Rules): void
- choice: int
# canPerfor
+ performM
KnightCharacter
+ activate(Rules, Game): void
+ deactivate(Rules, Game): void
PostmanChar
+ deactivate(Rules, Gar
+ activate(Rules, Game

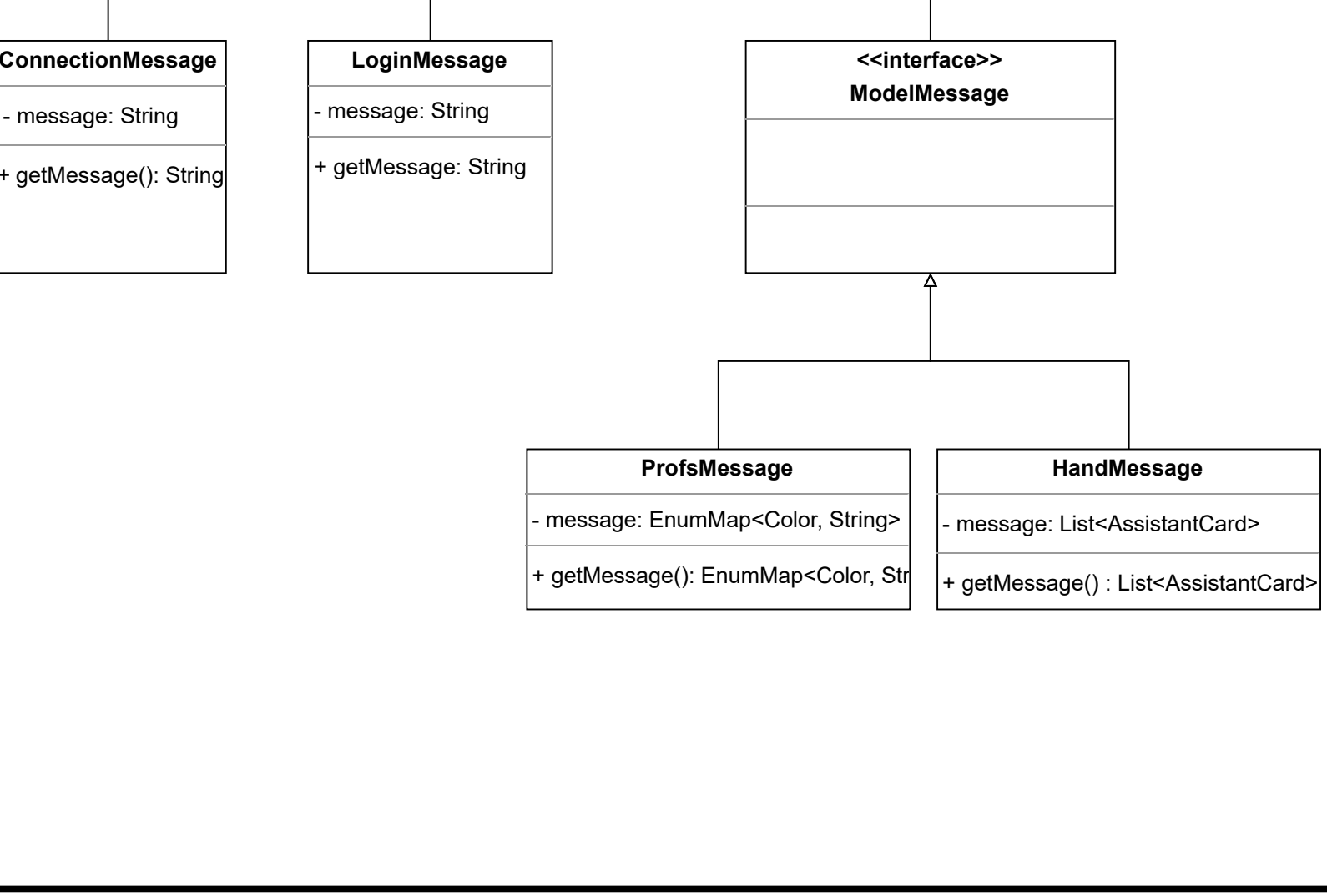

+ canPerform(Game, Rules): boolean
+ getNickname(): String
+ getPlayer(Game): Player
+ performMove(Game, Rules): void
PlayCard
rm(Game, Rules): void
ove(Game, Rules): void
<<abstract>>
MoveStudentFromEntry
- color: Color
# canPerform(Game, Rules): boolean
MoveStudentFromEntryToIsland
+ islandIndex: int
+ performMove(Game, Rules): void
MoveStudentFromEntryToHall
+ performMove(Game, Rules): void
racter
me): void
): void
<<interface>>
implements Serializable
Card

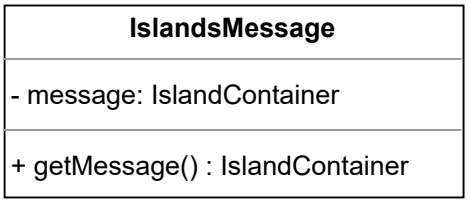


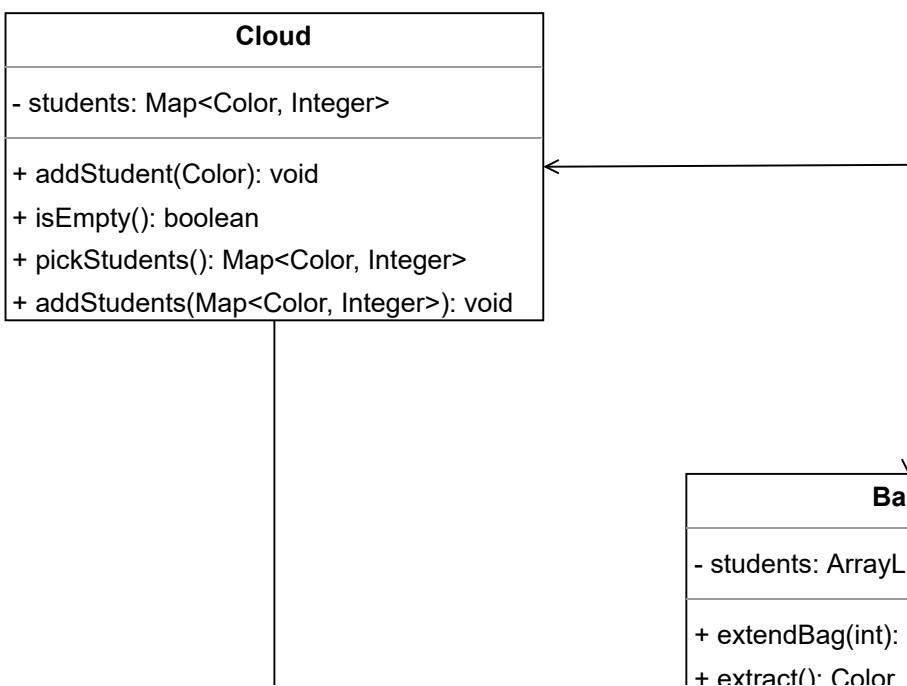

- clientToConnection: Map<VirtualClient, SocketClientCor
- currentGame: GameHandler
+ registerNewConnection(String, SocketClientConnection
+ unregisterClient(int): void
+ onMessaggeReceived(Message): void
+ lobby(SocketClientConnection)
SocketServer
- port: int
- server : Server
+ run(): void
+ acceptConnections(ServerSocket)

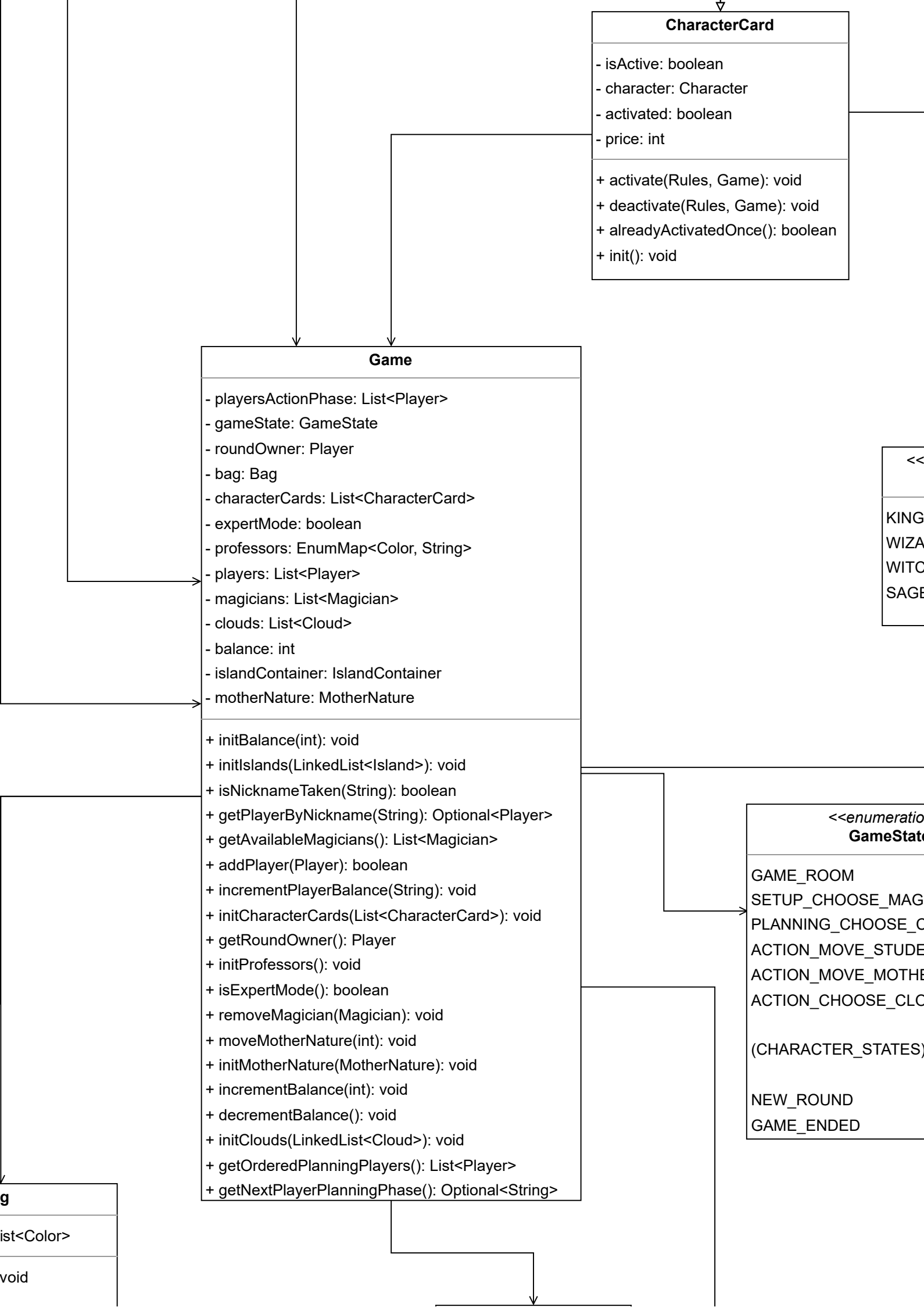

SocketClientConnection
- socket: Socker
- server: Server
- inputStream: ObjectInputStream
- outputStream: ObjectOutputStream
- clientID: Integer
- serverSocket: ServerSocket
+ close(): void
+ readFromStream(): void
+ run(): void
+ actionHandler(Message) : void
+ setupConnection(LoginMessage): void
+ setup(): void
C
- serverPort: int
- serverAddress: St
- outputStream: Obj
+ setup(): void
+ send(Message): v
NETWORK

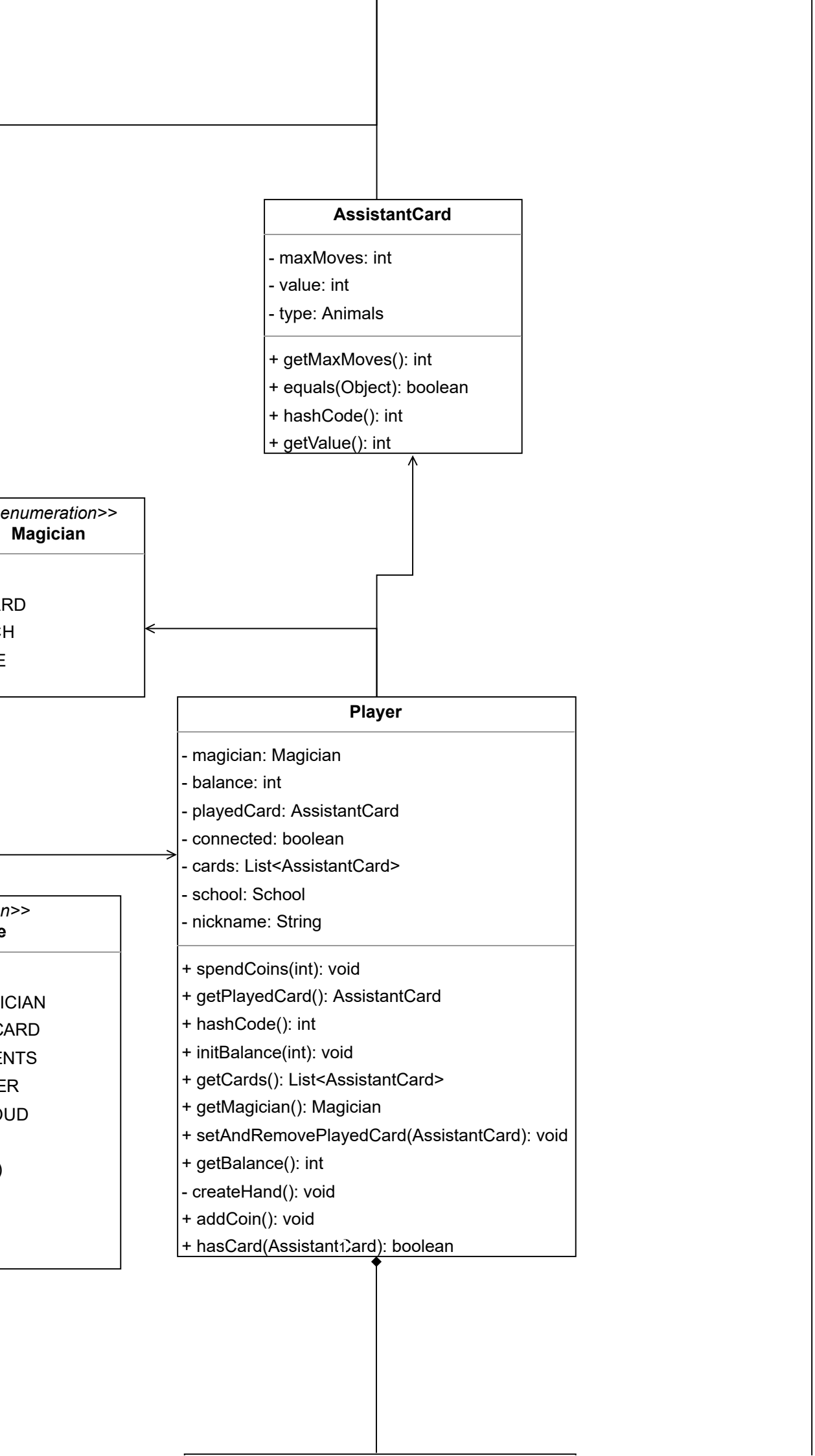

onnectionSocket
ring
jectOutputStream
void

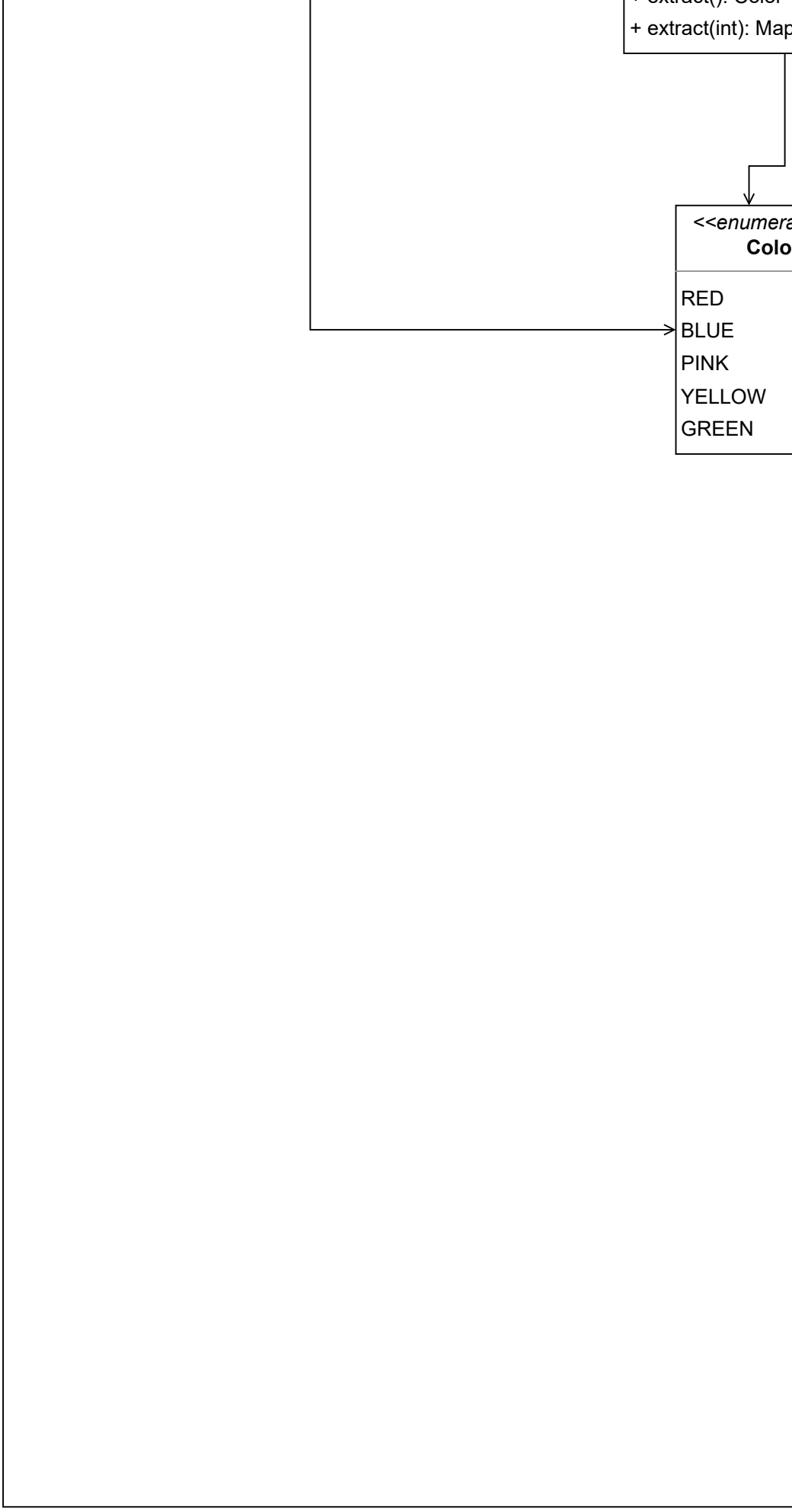

ConnectionMessage
- message: String
+ getMessage(): String
LoginMessage
- message: String
+ getMessage: String
<<interface>>
ModelMessage
ProfsMessage
- message: EnumMap<Color, String>
+ getMessage(): EnumMap<Color, Str
HandMessage
- message: List<AssistantCard>
+ getMessage() : List<AssistantCard>

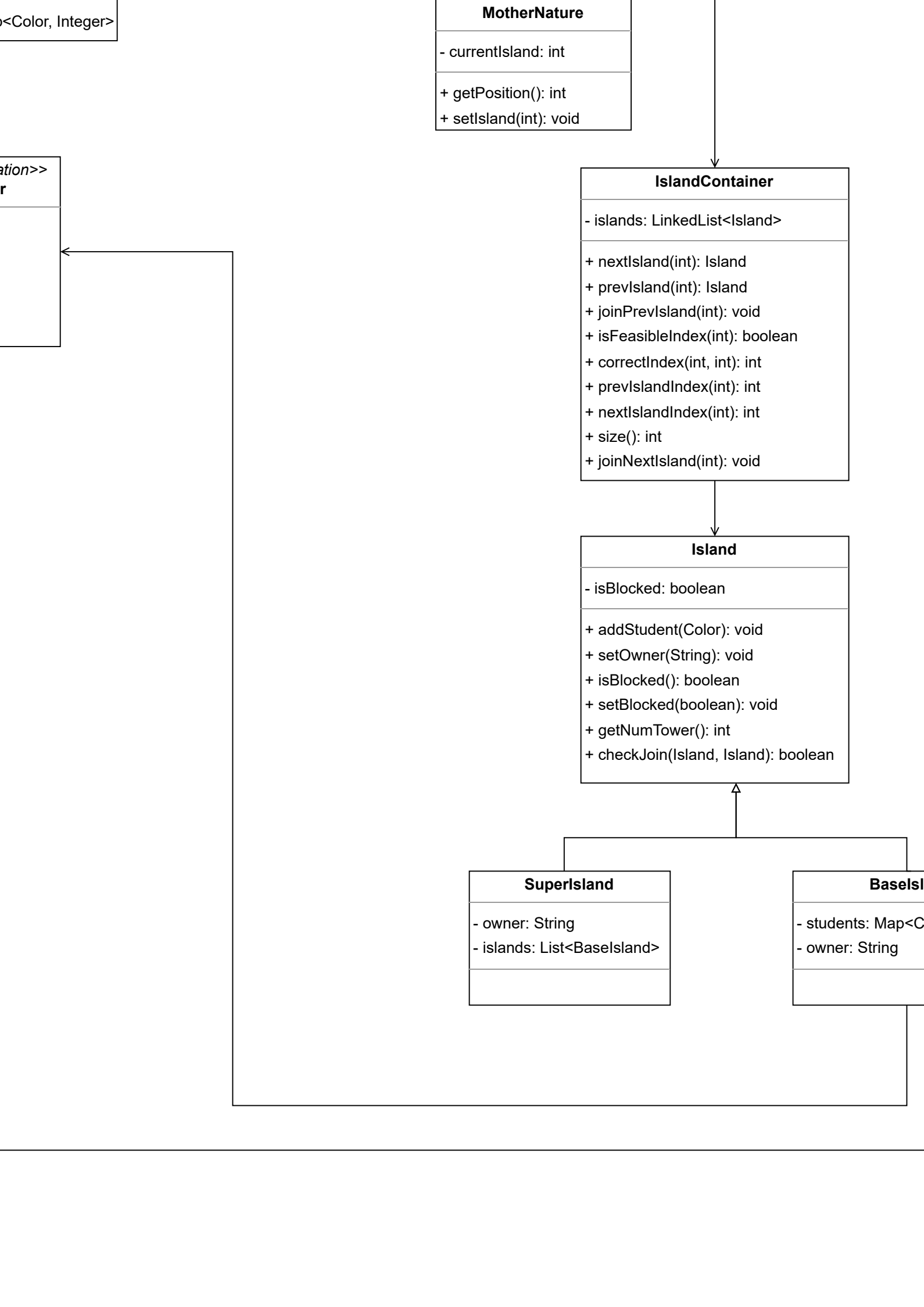

IslandsMessage
- message: IslandContainer
+ getMessage() : IslandContainer

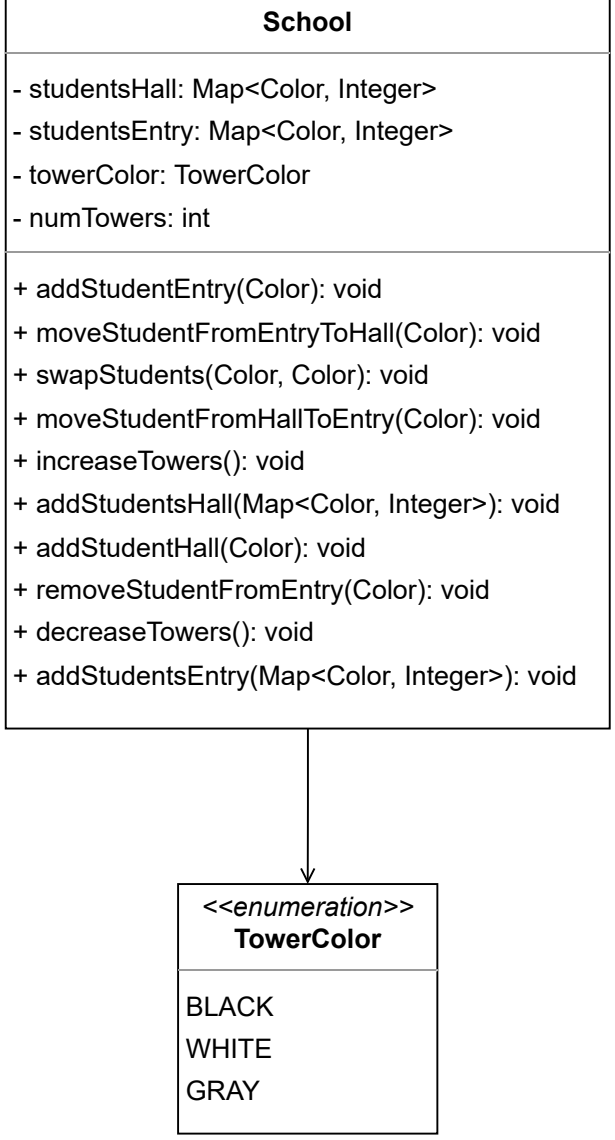

Cloud
- students: Map<Color, Integer>
+ addStudent(Color): void
+ isEmpty(): boolean
+ pickStudents(): Map<Color, Integer>
+ addStudents(Map<Color, Integer>): void
Ba
- students: ArrayL
+ extendBag(int):
+ extract(): Color


CharacterCard
- isActive: boolean
- character: Character
- activated: boolean
- price: int
+ activate(Rules, Game): void
+ deactivate(Rules, Game): void
+ alreadyActivatedOnce(): boolean
+ init(): void
Game
- playersActionPhase: List<Player>
- gameState: GameState
- roundOwner: Player
- bag: Bag
- characterCards: List<CharacterCard>
- expertMode: boolean
- professors: EnumMap<Color, String>
- players: List<Player>
- magicians: List<Magician>
- clouds: List<Cloud>
- balance: int
- islandContainer: IslandContainer
- motherNature: MotherNature
+ initBalance(int): void
+ initIslands(LinkedList<Island>): void
+ isNicknameTaken(String): boolean
+ getPlayerByNickname(String): Optional<Player>
+ getAvailableMagicians(): List<Magician>
+ addPlayer(Player): boolean
+ incrementPlayerBalance(String): void
+ initCharacterCards(List<CharacterCard>): void
+ getRoundOwner(): Player
+ initProfessors(): void
+ isExpertMode(): boolean
+ removeMagician(Magician): void
+ moveMotherNature(int): void
+ initMotherNature(MotherNature): void
+ incrementBalance(int): void
+ decrementBalance(): void
+ initClouds(LinkedList<Cloud>): void
+ getOrderedPlanningPlayers(): List<Player>
+ getNextPlayerPlanningPhase(): Optional<String>
<<
KING
WIZA
WITC
SAGE
<<enumeratio
GameStat
GAME_ROOM
SETUP_CHOOSE_MAG
PLANNING_CHOOSE_C
ACTION_MOVE_STUDE
ACTION_MOVE_MOTHE
ACTION_CHOOSE_CLO
(CHARACTER_STATES)
NEW_ROUND
GAME_ENDED
g
ist<Color>
void

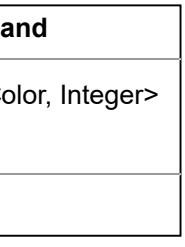

AssistantCard
- maxMoves: int
- value: int
- type: Animals
+ getMaxMoves(): int
+ equals(Object): boolean
+ hashCode(): int
+ getValue(): int
enumeration>>
Magician
RD
H
E
Player
- magician: Magician
- balance: int
- playedCard: AssistantCard
- connected: boolean
- cards: List<AssistantCard>
- school: School
- nickname: String
+ spendCoins(int): void
+ getPlayedCard(): AssistantCard
+ hashCode(): int
+ initBalance(int): void
+ getCards(): List<AssistantCard>
+ getMagician(): Magician
+ setAndRemovePlayedCard(AssistantCard): void
+ getBalance(): int
- createHand(): void
+ addCoin(): void
+ hasCard(AssistantCard): boolean
n>>
e
ICIAN
CARD
ENTS
ER
OUD
)


+ extract(int): Map
<<enumera
Colo
RED
BLUE
PINK
YELLOW
GREEN


o<Color, Integer>
MotherNature
- currentIsland: int
+ getPosition(): int
+ setIsland(int): void
ation>>
r
IslandContainer
- islands: LinkedList<Island>
+ nextIsland(int): Island
+ prevIsland(int): Island
+ joinPrevIsland(int): void
+ isFeasibleIndex(int): boolean
+ correctIndex(int, int): int
+ prevIslandIndex(int): int
+ nextIslandIndex(int): int
+ size(): int
+ joinNextIsland(int): void
Island
- isBlocked: boolean
+ addStudent(Color): void
+ setOwner(String): void
+ isBlocked(): boolean
+ setBlocked(boolean): void
+ getNumTower(): int
+ checkJoin(Island, Island): boolean
SuperIsland
- owner: String
- islands: List<BaseIsland>
BaseIsl
- students: Map<C
- owner: String


School
- studentsHall: Map<Color, Integer>
- studentsEntry: Map<Color, Integer>
- towerColor: TowerColor
- numTowers: int
+ addStudentEntry(Color): void
+ moveStudentFromEntryToHall(Color): void
+ swapStudents(Color, Color): void
+ moveStudentFromHallToEntry(Color): void
+ increaseTowers(): void
+ addStudentsHall(Map<Color, Integer>): void
+ addStudentHall(Color): void
+ removeStudentFromEntry(Color): void
+ decreaseTowers(): void
+ addStudentsEntry(Map<Color, Integer>): void


<<enumeration>>
TowerColor
BLACK
WHITE
GRAY


and
olor, Integer>