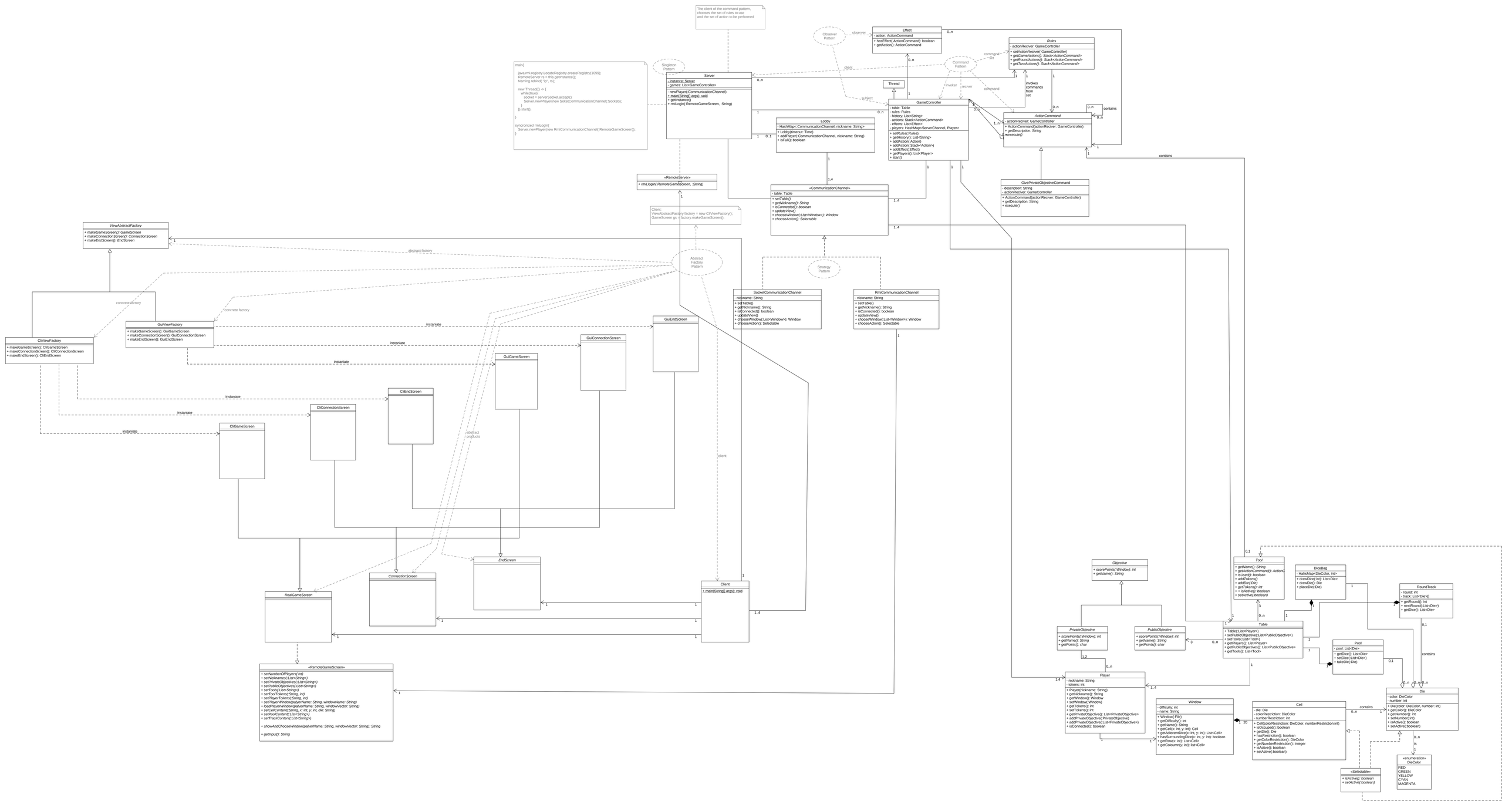
The client of the command pattern, chooses the set of rules to use and the set of action to be performed
Observer Pattern
observer
Effect
- action: ActionCommand
+ hasEffect(:ActionCommand): boolean
+ getAction(): ActionCommand
0..n
Rules
- actionReciver: GameController
+ setActionReciver(:GameController)
+ getGameActions(): Stack<ActionCommand>
+ getRoundActions(): Stack<ActionCommand>
+ getTurnActions(): Stack<ActionCommand>
command set
Command Pattern
client
invoker
reciver
command
invokes commands from set
main{
java.rmi.registry.LocateRegistry.createRegistry(1099);
RemoteServer rs = this.getInstance();
Naming.rebind("gc", rs);
new Thread(() -> {
while(true){
socket = serverSocket.accept()
Server.newPlayer(new SoketCommunicationChannel(:Socket));
}
}).start();
}
syncronized rmiLogin{
Server.newPlayer(new RmiCommunicationChannel(:RemoteGameScreen));
}
Singleton Pattern
Server
- instance: Server
- games: List<GameController>
- newPlayer(:CommunicationChannel)
+ main(String[],args):L.void
+ getInstance()
+ rmiLogin(:RemoteGameScreen, :String)
0..n
Thread
subject
GameController
- table: Table
- rules: Rules
- history: List<String>
- actions: Stack<ActionCommand>
- effects: List<Effect>
- players: HashMap<ServerChannel, Player>
+ setRules(:Rules)
+ getHistory(): List<String>
+ addAction(:Action)
+ addActions(:Stack<Action>)
+ addEffect(:Effect)
+ getPlayers(): List<Player>
+ start()
ActionCommand
- actionReciver: GameController
+ ActionCommand(actionReciver: GameController)
+ getDescription :String
#execute()
contains
Lobby
- HashMap<:CommunicationChannel, nickname: String>
+ Lobby(timeout: Time)
+ addPlayer(:CommunicationChannel, nickname: String)
+ isFull(): boolean
«RemoteServer»
+ rmiLogin(:RemoteGameScreen, :String)
«CommunicationChannel»
- table: Table
+ setTable()
+ getNickname(): String
+ isConnected(): boolean
+ updateView()
+ chooseWindow(:List<Window>): Window
+ chooseAction(): Selectable
Client:
ViewAbstractFactory factory = new CliViewFactory();
GameScreen gs = factory.makeGameScreen();
GiveMePrivateObjectiveCommand
- description: String
- actionReciver: GameController
+ ActionCommand(actionReciver: GameController)
+ getDescription: String
+ execute()
ViewAbstractFactory
+ makeGameScreen(): GameScreen
+ makeConnectionScreen(): ConnectionScreen
+ makeEndScreen(): EndScreen
abstract factory
Abstract Factory Pattern
Strategy Pattern
concrete factory
concrete factory
GuiViewFactory
+ makeGameScreen(): GuiGameScreen
+ makeConnectionScreen(): GuiConnectionScreen
+ makeEndScreen(): GuiEndScreen
CliViewFactory
+ makeGameScreen(): CliGameScreen
+ makeConnectionScreen(): CliConnectionScreen
+ makeEndScreen(): CliEndScreen
instaniate
instaniate
instaniate
instaniate
instaniate
instaniate
GuiEndScreen
GuiConnectionScreen
GuiGameScreen
CliEndScreen
CliConnectionScreen
CliGameScreen
abstract products
client
SocketCommunicationChannel
- nickname: String
+ setTable()
+ getNickname(): String
+ isConnected(): boolean
+ updateView()
+ chooseWindow(:List<Window>): Window
+ chooseAction(): Selectable
RmiCommunicationChannel
- nickname: String
+ setTable()
+ getNickname(): String
+ isConnected(): boolean
+ updateView()
+ chooseWindow(:List<Window>): Window
+ chooseAction(): Selectable
EndScreen
ConnectionScreen
RealGameScreen
Client
+ main(String[],args):L.void
«RemoteGameScreen»
+ setNumberOfPlayers(:int)
+ setNicknames(:List<String>)
+ setPrivateObjectives(:List<String>)
+ setPublicObjectives(:List<String>)
+ setTools(:List<String>)
+ setToolTokens(:String, int)
+ setPlayerTokens(:String, int)
+ setPlayerWindow(:palyerName: String, windowName: String)
+ loadPlayerWindow(:palyerName: String, windowVector: String)
+ setCellContent(:String, x: int, y: int, die: String)
+ setRoolContent(:List<String>)
+ setTrackContent(:List<String>)
+ showAndChooseWindow(:palyerName: String, windowVector: String): String
+ getInput(): String
Objective
+ scorePoints(:Window): int
+ getName(): String
PrivateObjective
+ scorePoints(:Window): int
+ getName(): String
+ getPoints(): char
PublicObjective
+ scorePoints(:Window): int
+ getName(): String
+ getPoints(): char
Player
- nickname: String
- tokens: int
+ Player(nickname: String)
+ getNickname(): String
+ getWindow(): Window
+ setWindow(:Window)
+ getTokens(): int
+ setTokens(): int
+ getPrivateObjective(): List<PrivateObjective>
+ addPrivateObjective(:PrivateObjective)
+ addPrivateObjective(:List<PrivateObjective>)
+ isConnected(): boolean
Tool
+ getName(): String
+ getActionCommand(): ActionC
+ isUsed(): boolean
+ addTokens()
+ addDie(:Die)
+ getTokens(): int
+ = isActive(): boolean
+ setActive(:boolean)
DiceBag
- HaltsMap<DieColor, int>
+ drawDice(:int): List<Die>
+ drawDie(): Die
+ placeDie(:Die)
RoundTrack
- round: int
- track: List<Die>[]
+ getRound(): int
+ nextRound():List<Die>)
+ getDice(): List<Die>
Table
+ Table(:List<Player>)
+ setPublicObjective(:List<PublicObjective>)
+ setTools(:List<Tool>)
+ getPlayers(): List<Player>
+ getPublicObjectives(): List<PublicObjective>
+ getTools(): List<Tool>
Pool
- pool: List<Die>
+ getDice(): List<Die>
+ setDice(:List<Die>)
+ takeDie(:Die)
contains
Window
- difficulty: int
- name: String
+ Window(:File)
+ getDifficulty(): int
+ getName(): String
+ getCell(x: int, y: int): Cell
+ getAdiacentDice(x: int, y: int): List<Cell>
+ hasSurroundingDice(x: int, y: int): boolean
+ getRow(x: int): List<Cell>
+ getColoumn(y: int): list<Cell>
Cell
- die: Die
- colorRestriction: DieColor
- numberRestriction: int
+ Cell(colorRestriction: DieColor, numberRestriction:int)
+ isOccupied(): boolean
+ getDie(): Die
+ hasRestriction(): boolean
+ getColorRestriction(): DieColor
+ getNumberRestriction(): Integer
+ isActive(): boolean
+ setActive(:boolean)
Die
- color: DieColor
- number: int
+ Die(color: DieColor, number: int)
+ getColor(): DieColor
+ getNumber(): int
+ setNumber(:int)
+ isActive(): boolean
+ setActive(:boolean)
«Selectable»
+ isActive(): boolean
+ setActive(:boolean)
«enumeration»
DieColor
RED
GREEN
YELLOW
CYAN
MAGENTA