Lab ex 7

del Castillo, Kyle Adrian
Dy, Alwyn

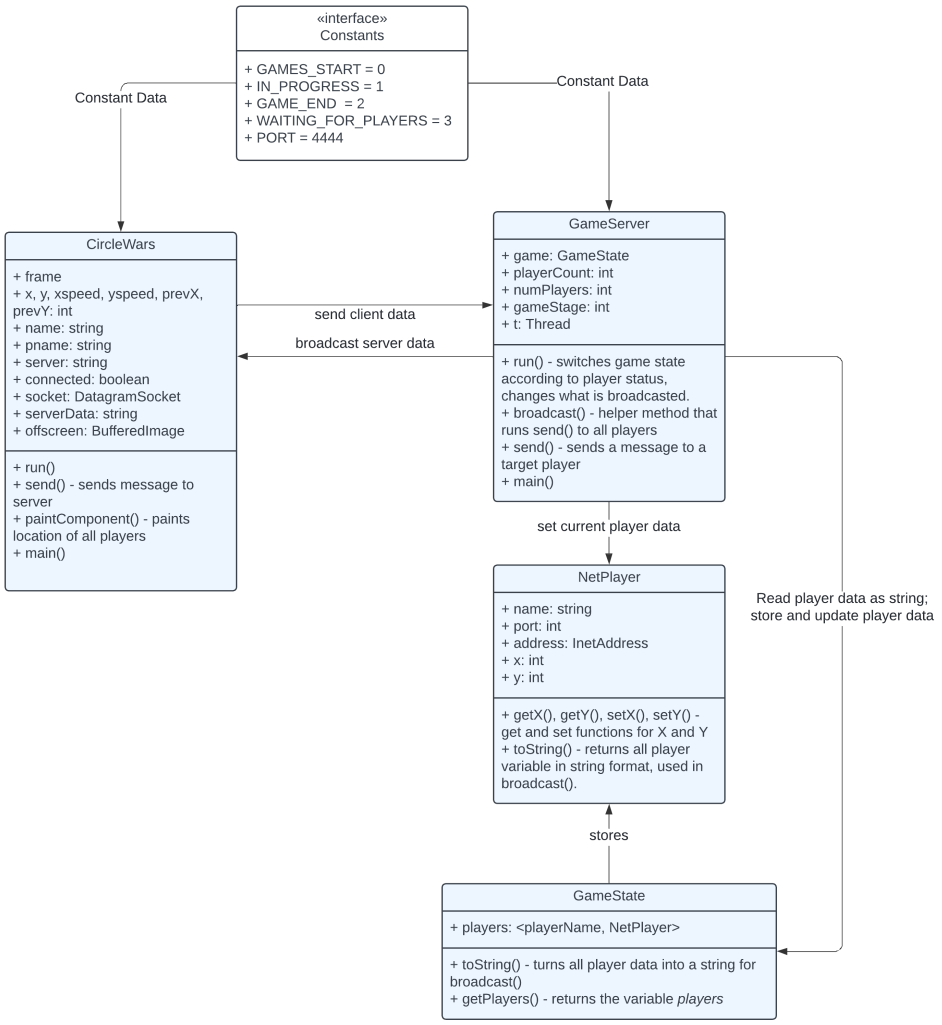
1. What are the different classes involved? (HINT: There are 5 of them)
    - CircleWars
    - GameServer
    - GameState
    - NetPlayer
    - Constants
2. For each class, list all the attributes and methods.

Class	Attributes	Methods
CircleWars	frame, t, name, pname, server, connected, socket, serverData, offscreen	CircleWars (contructor), main, send, run, paintComponent
GameServer	playerData, playerCount, serverSocket, game, gameStage, numPlayers, t	GameServer (constructor), main, broadcast, send, run
GameState	players	GameState (constructor), update, toString, getPlayers
NetPlayer	address, port, name, x, y	NetPlayer (constructor), getAddress, getPort, getName, setX, getX, getY, setY, toString
Constants	APP_NAME, GAME_START, IN_PROGRESS, GAME_END, WAITING_FOR_PLAYERS, PORT	

3. What does each class perform?
    - **CircleWars**
        - Handles the client functionalities
        - Finds and connects to the server
            - Requests connection to the server following the format `CONNECT <player_name>` along with the client's address and port
            - Receives confirmation of connection from the server following the format `CONNECTED <player_name>`
        - Receives data from server containing locations of all players
            - Data of each player is in the format `PLAYER <player_name> <x> <y>`
            - Data of multiple players are separated by `:` e.g. `PLAYER <player_name> <x> <y>:PLAYER <player_name> <x> <y>`
        - Draws (or paints) a circle at location of all players transmitted by the server
        - Handles and translates the mouse movements or arrow key presses to movements of the player's circle
        - Transmits the location of the player's circle in the client instance to the server following the format `PLAYER <player_name> <x> <y>`
    - **NetPlayer**
        - Stores the information of a player
            - Address
            - Port number
            - Name
            - X and Y location
        - Handles the update of the X and Y location of a player in the game
        - Returns the current X and Y location of a player
        - Converts information of a player to string following the format `PLAYER <player_name> <x> <y>`
    - **GameState**
        - Stores and handles updates to the game state including the players in the game (NetPlayer)
        - Converts the current game state into a string for broadcast
    - **GameServer**
        - Handles game server functionalities
        - Listens to a specific port (Port 4444)
        - Initializes and handles connection to and from the clients
            - Receives connection request from clients following the format `CONNECT <player_name>`
            - Sends confirmation of connection to the client following the format `CONNECTED <player_name>`
        - Monitors stage of the game, either waiting for players, game is starting, or game is in progress
            - Game only starts when the specified number of players is reached

        - Receives packets transmitted by the clients following the format `PLAYER <player_name> <x> <y>`
        - Updates the game state with the location from the packet
        - Broadcasts the updated game state to all clients
    - **Constants**
        - Stores the constants used in the game including the app name (used in the client's window title), port number (listened by the server), and the int representation of the game stage

4. How does each class interact and relate with the other classes? Show a diagram that depicts these interactions and relationships among the different classes


«interface»
Constants
+ GAMES_START = 0
+ IN_PROGRESS = 1
+ GAME_END = 2
+ WAITING_FOR_PLAYERS = 3
+ PORT = 4444
Constant Data
Constant Data
CircleWars
+ frame
+ x, y, xspeed, yspeed, prevX, prevY: int
+ name: string
+ pname: string
+ server: string
+ connected: boolean
+ socket: DatagramSocket
+ serverData: string
+ offscreen: BufferedImage
+ run()
+ send() - sends message to server
+ paintComponent() - paints location of all players
+ main()
send client data
broadcast server data
GameServer
+ game: GameState
+ playerCount: int
+ numPlayers: int
+ gameStage: int
+ t: Thread
+ run() - switches game state according to player status, changes what is broadcasted.
+ broadcast() - helper method that runs send() to all players
+ send() - sends a message to a target player
+ main()
set current player data
NetPlayer
+ name: string
+ port: int
+ address: InetAddress
+ x: int
+ y: int
+ getX(), getY(), setX(), setY() - get and set functions for X and Y
+ toString() - returns all player variable in string format, used in broadcast().
Read player data as string; store and update player data
stores
GameState
+ players: <playerName, NetPlayer>
+ toString() - turns all player data into a string for broadcast()
+ getPlayers() - returns the variable *players*

    -

5. How the exchange of messages between the client and the server classes (HINT: see the send() and receive() methods) for (a) Starting the game, (b) game proper, and (3) ending the game.
    - The communication between the client (`CircleWars`) and the server (`GameServer`) appears to happen through the `send()` and `run()` methods on the client side, and `receive()` and `broadcast()` methods on the server side.
    - **Starting the Game**
        - *Client (`CircleWars`) Side:**
            - when the client starts, it sends a "`CONNECT <name>`" message to the server
        - *Server (`GameServer`) Side*
            - the `run()` method checks for the "`CONNECT`" message when waiting for the players

```
if (playerData.startsWith("CONNECT")) {
    // Process and add the player to the game state
    // Broadcast the connection to all players
    broadcast("CONNECTED " + tokens[1]);
    playerCount++;
    // If enough players are connected, transition to the GAME_START state
    if (playerCount == numPlayers) {
        gameStage = GAME_START;
        }
  }
```

- **Game Proper**
  - *Client (`CircleWars`) Side*
    - the client continuously sends `"PLAYER <name> <x> <y>"` messages as the player moves.
  - *Server (`GameServer`) Side*
    - In the `run()` method during the `IN_PROGRESS` state, the server checks for `"PLAYER"` messages and updates the game state

```
if (playerData.startsWith("PLAYER")) {
        // tokenize player data update and extract
        // get old player data from gamestate and update
        broadcast(game.toString());
}
```

- **Ending the Game**
  - there was no explicit handling of the game end, nor does there appear to be any rules designed around ending the game.