2.2.1 - Package common

2.2.1.1 - Message

- Quoi? Un message échangé entre un serveur et un client (à travers une socket)
- Constructeur: `Message(idSender, type, content)`
- ---------- Variables ----------
  - `int idSender` : l'id de l'envoyeur
  - `TypeMsg type` : le type de message (qui est définit juste en dessous)
  - `enum TypeMsg` :
    - Quoi? Le type de message envoyé, 7 types:
      - `NAME` : Le client a envoyé son nom via la fenêtre d'identification.
      - `ID` : Le serveur retourne l'identifiant qu'il a associé au client.
      - `RECHERCHE` : Le client a cliqué sur un bouton de recherche de jeu (ex: "3+0")
      - `STARTGAME` : Le serveur ordonne aux deux clients de commencer la partie.
      - `MP` : Un joueur a envoyé un message perso à son adversaire.
      - `REQUETE` : Un joueur demande l'autorisation au serveur de jouer un coup.
      - `ORDRE` : Le serveur autorise le coup et ordonne à chaque clients de mettre à jour leur interface pour afficher le coup joué.
  - `content` :
    - Quoi? le contenu du message, example : *"content"* (`TypeMsg`)
      - *"Fred"* (`NAME`)
      - *"2"* (`ID`)
      - *"3+0"* (`RECHERCHE`)
      - *"Pueblo/black"* (`STARTGAME`) :
        - Syntax: *"nom_adversaire/couleur_adversaire"* : permet aux clients de connaître leurs couleurs et le nom de l'adversaire.
      - *"Good luck"* (`MP`)
      - Syntax pour `REQUETE` & `ORDRE` : *"PieceToRemove/FromThisSquare/PieceToAdd/ToThisSquare"*
        - Ex1: *"whiteBishop/9/whiteBishop/2"*
        - Ex2: Promotion: *"blackPawn/51/blackQueen/59"*
          1. Remove the black pawn
          2. from the C7 square (1=A1, 8=A8, 9=B1, etc..)
          3. and print a black queen

4. on the C8 square

- ---------- Methods ----------
  - `TypeMsg getType()` : retourne le `type` du `Message`
  - `void setId(int idSender)` : affecter la variable `idSender` avec la valeur du client qui a envoyé le message.
  - `int getIdSender()` : retourne l'id de l'envoyeur
  - `String getContent()` : retourne le contenu du message
  - `void printMsg()` : string representation of the message

2.2.2 - Package server

2.2.2.1 - MainServer

- Quoi? Créer le server à partir du port
- ---------- Methods ----------
  - `public static void main()`
  - `void printUsage()` : print message d'erreur si mauvais arguments rentrés

2.2.2.2 - Server

- Quoi? Le serveur de l'application. Responsable des `Clients` et les `Game`
- Constructeur : `Server(port)`
  => Créer et lance un thread `Connection`
- ---------- Variables ----------
  - `int port` : Port sur lequel le server écoute les nouvelles connexions
  - `List<ConnectedClient> clients` : Liste des clients connectés
  - `List<Game> games` : Liste des parties crées
- ---------- Methods ----------
  - `void addClient()` : Ajoute le `ConnectedClient` à la liste de `clients`
  - `void createGame(ConnectedClient client, String format)` : Créer un objet `Game()` et y associe un premier joueur
  - `searchGame(ConnectedClient client, String format)`
    1. Recherche parmi les `games` dont `Game.players.size()` < 2 si il en existe avec ce `format`
       1. Si oui, appel `Game.addPlayerB(client)` puis `Game.startGame()`
       2. Si non, appel `createGame(client, format)`
  - `disconnectedClient(ConnectedClient client)` : Appelle `Client.closeClient()` + le retire de la liste `clients`
  - `getPort()` : retourne le port du serveur

2.2.2.3 - Connection

- Quoi? Thread gérant les connections de nouveau clients au serveur
- Constructeur : `Connection(Server server)`
  => Ouvre une socket sur le port reçu du server
- ---------- Variables ----------
  - `Server server` : référence vers le serveur qui crée cette connection
  - `ServerSocket serversocket` : créer et ouvre une socket qui écoute sur le port
- ---------- Methods ----------
  - `run()` : lancé par le server (`.start()`)
    1. Accept les nouvelles connections (`.accept()`)
    2. Instancie un `ConnectedClient` et appelle `Server.addClient()`
    3. Lance un thread sur ce `ConnectedClient` (`.start()`)

2.2.2.4 - ConnectedClient

- Quoi? Un client connecté au server
- Constructeur : `ConnectedClient(Server server, Socket socket)`
  => Instancie les variables du client et créer deux streams permettant la communication avec le server
- ---------- Variables ----------
  - `int nbClients` : compteur d'instance (de `ConnectedClient`) crées
  - `int id` : id du client (= `nbClients`)
  - `String name` : nom du client
  - `Server server` : référence vers le serveur
  - `Socket socket` : référence vers la socket du client (qui est lié avec la socket du server `serverSocket` puisque je l'ai `.accept()`)
  - `ObjectOutputStream out` : va permettre d'envoyer des messages au client (en utilisant le flux de sortie de sa socket)
  - `ObjectInputStream in` : recevoir des messages du client
  - `Game game` : la partie associée au joueur (si existe)
- ---------- Methods ----------
  - `run()` : thread (lancé dans `Connection`) qui guette constamment les incoming `Message` du client (`in.readObject()`) et les traitent accordingly
    - Si type= `NAME` : appelle `ConnectedClient.setName()`
    - Si type= `MP` : retransmet le `Message` au `ConnectedClient` destinataire (adversaire) via `Game.sendMP()`
    - Si type= `RECHERCHE` : le client veut jouer. Appelle `Server.searchGame()`

        - Si type=`REQUETE` : le client demande à jouer un coup. Appelle `Game.playMove()`
    - `sendMessage(Message mess)` : envoie le `Message` au client
    - `closeClient()` : Ferme les flux in/out + la socket
    - `setName()`
    - `getName()`
    - `getId()`
    - `void setGame(Game game)` : Assigne un `Game` au présent `ConnectedClient`

2.2.2.5 - MainBDD

- Quoi? La connection avec la BDD
- ---------- Variables ----------
    - `String URL`
    - `String USER`
    - `String PASSWORD`
- ---------- Methods ----------
    - `void main(String[] args)`
    - `Connection getConnection()`
    - `void addUser(String nom, String email)` : Envoie query pour ajouter le client à la BDD

2.2.2.6 - Game

- Quoi? Représente une partie d'échec. Stock toutes les informations de la partie et gère la logique de déplacements. Renverra tout ordre vers `GamePane` (coté client)
- Constructeur : `Game(ConnectedClient playerA, String format)`
  => instancie les variables du game et ajoute le `ConnectedClient` à `players`
- ---------- Variables ----------
    - `ConnectedClient playerA, playerB` : le client ayant demandé... `A` => la création du game `B` => à rejoindre le Game existant
    - `ConnectedClient players` : liste des joueurs (deux au maximum)
    - `String couleurA, couleurB` : couleur des joueurs
    - `String format` : le format du jeu (ex: "3+2")
    - `boolean hasStart` : la partie a-t-elle commencé ?
    - `HashMap<Integer, String> piecePositions` : pour chaque 64 cases, le nom de la pièce qui s'y trouve (si existe)
- ---------- Methods ----------
    - `sendMP(Message mess)` : Envoie (retransmet) le message reçu du playerA (dans `ConnectedClient.run()`) au playerB

- `startGame()` : appelée dans `Server.searchGame()`. Défini qui est noir/blanc, appelle `setInitialPiecePositions()` et renvoie un `Message` de type "STARTGAME" aux joueurs
- `setInitialPiecePositions()` : Modifie `piecePositions` pour qu'il reflète la position des pièces au début d'une partie.
- `playMove(String requestPlayer, ConnectedClient playerwhoPlayedTheMove)` : gère la logique de déplacement des pièces (Voir `Message` pour le format du message).
- `boolean checkMove(String requestPlayer, ConnectedClient playerwhoPlayedTheMove)` : Autorise ou non le coup passé en paramètre (requested par le joueur)
- `addPlayerB(ConnectedClient playerB)` : ajoute un 2ème clients à `players`
- `List<ConnectedClient> getPlayers()` : return `players`
- `String getFormat()` : return `format`
- `boolean gameHasStarted()` : return `hasStart`
- `String getColorClient(ConnectedClient client)` : retourne la couleur d'un client.

2.2.3 - Package client

2.2.3.1 - MainClient

- Quoi? Créer la fenêtre de l'application, lance un client, s'ouvre sur identification. Un seul `Stage` sera créer auquel on attachera et changera les `Scene` (`IdentificationPanel`, `MenuPanel`, `GamePanel`)
- ---------- Variables ----------
  - `Stage stage`
  - `Client client` : le client
  - `Group root` : groupe racine auquel on ajoutera `identificationPanel`
  - `StackPane root` : le groupe utilisé pour les scene `MenuPanel` et `GamePanel`
  - `IdentificationPanel identificationPanel` : 1ère scène : identification => S'affiche à l'instanciation de cette présente classe.
  - `MenuPanel menuPanel` : 2ème scène : menu => S'affichera une fois que le client aura envoyé son nom via `identificationPanel.sendBtn.setOnAction()
  - `GamePanel gamePanel` : 3ème scène : la partie => S'affichera une fois que le client aura cliqué sur un des boutons via `menuPanel.btn_x_x_setOnAction()`.
  - `double screen_width, screen_height, scene_height, scene_width`
  - `double stageX, stageY`
- ---------- Methods ----------
  - `void main()` : lance la fenêtre `Application.launch()`

- `void start(Stage stage)` : créer un `Client` avec l'addresse et le port et créer et affiche la 1ère scène : `identificationPanel`
- `void switchSceneToMenu()` : Affiche la 2ème scène
- `void switchSceneToGame(String format)` : Affiche la 3ème scène. Appelle `gamePanel.setUpBoard()` et `gamePanel.setProperties()` ainsi que `client.setGamePanel()`
- `interface SceneSwitchListener` : Interface permettant d'executer les méthodes qu'elle contient dans d'autre classes (`IdentificationPanel` et `MenuPanel`)

2.2.3.2 - Client

- Quoi? Un client qui se connectera au serveur
- Constructeur : `Client(address, port)`
  => Initialise les variables du clients, créer une socket et un stream in & out pour la communication avec le server et créer et `.start()` le thread `ClientReceive`
- ---------- Variables ----------
  - `String address` : IP server
  - `int port` : port server
  - `String name` : nom client
  - `int id` : id client
  - `Socket socket` : socket pour communiquer avec le server
  - `ObjectInputStream in` : pour recevoir des messages du server
  - `ObjectOutputStream out` : pour envoyer des messages au server
  - `GamePanel gamePanel` : la scène `GamePanel` du client
- ---------- Methods ----------
  - `void messageReceived(Message mess)` : Handle server's messages (types: `ID` , `ORDRE` , `STARTGAME` , `MP`)
  - `sendMessage(Message mess)` : Envoie `mess` au `Server`
  - `getObjInputStream()` : return the input stream
  - `void setName(String name)` : Assigne le nom du joueur
  - `String getName()`
  - `setId(int id)`
  - `int getId()`
  - `void setGamePanel(GamePanel gamepane)`

2.2.3.3 - ClientReceive

- Quoi? Thread permettant la réception de message du server
- Constructeur : `ClientReceive(client)`

- ---------- Variables ----------
  - `Client client`
  - `ObjectInputStream in`
- ---------- Methods ----------
  - `run()` : thread (lancé dans `Client`) qui guette constamment les incoming `Message` du server (`in.readObject()`) et les envoient à `Client.messageReceived()`

2.2.3.4 - IdentificationPanel

- Quoi? La vue d'identification qui invitera le client à saisir son nom
- Constructeur : `IdentificationPanel(Client client, SceneSwitchListener sceneSwitchListener)`
  => Créer et set le champ de saisie et le bouton d'envoie.
- ---------- Variables ----------
  - `TextArea playerNameToSend` : le champ de saisie
  - `Button sendBtn`
  - `Client client`
  - `SceneSwitchListener sceneSwitchListener` : interface dans `MainClient`, permettant d'executer les méthodes `MainClient.switchSceneTo...()`
- ---------- Methods ----------
  - `sendBtn.setOnAction()` : Envoie du contenu de `playerNameToSend` via `client.sendMessage()` + stocke dans la BDD

2.2.3.5 - MenuPanel

- Quoi? Le menu principal qui s'affiche une fois que le client a envoyé son nom via `IdentificationPanel`
- Constructeur : `MenuPanel(Client client, SceneSwitchListener sceneSwitchListener, double scene_width, double scene_height)`
  => Créer et set les boutons pour lancer une partie
- ---------- Variables ----------
  - `BorderPane MainPane` : parent component qui contiendra tous les autres composants
  - `StackPane root` : le groupe racine de `MainClient`
  - `GridPane gridFormats` : grille 3x3 contient les formats
  - `Button[] buttons`
  - `Client client`
  - `SceneSwitchListener sceneSwitchListener` : interface dans `MainClient`, permettant d'executer les méthodes `MainClient.switchSceneTo...()`
- ---------- Methods ----------

- `createButton(String text, int columnIndex, int rowIndex)` : appelé dans le constructeur pour instancier les boutons.
- `button.setOnAction()` : Envoie message (ex: content="3+1") via `Client.sendMessage()`
- `void setProperties(StackPane root)`

2.2.3.6 - GamePanel

- Quoi? Un plateau d'échec avec les pièces, un timer, le nom de l'adversaire, un chat
- Constructeur : `GamePanel(Client client, SceneSwitchListener sceneSwitchListener , double sceneWidth, double sceneHeight, String format)`
- ---------- Variables ----------
  - `SceneSwitchListener sceneSwitchListener`
  - `StackPane root`
  - `Client client`
  - `HBox mainPane` : parent component qui contiendra tous les autres composants de la scène
  - `String imagePath`
    => PLATEAU
  - `GridPane boardPane` : plateau composé de 8x8 cases
  - `Paint[][] colorSquares` : couleur des cases
  - `int[] firstClick = new int[2]`
  - `ImageView selectedPiece` : pièce sélectionnée par le joueur
  - `HashMap<Integer, String> piecePositions` : position pièce => nom pièce
    => RIGHT : CHAT + TIMER & NAME
  - `BorderPane rightPane`
    => TIMER & NAME
  - `HBox playerAPane` : contains the name of the playerA + its timer.
    - `Text namePlayerA`
    - `Text timerPlayerA`
  - `HBox playerBPane` : contains the name of the playerB + its timer.
    - `Text namePlayerB`
    - `Text timerPlayerA`
    => CHAT
  - `BorderPane chatPane` : le chat
  - `ScrollPane scrollReceivedText`
  - `TextFlow receivedText`
  - `HBox sendBox`

- `TextArea textToSend` : saisie text
- `Button sendBtn` : envoie du message
  => AUTRE
- `String format` : format de jeu
- `String namePlayerAStr`
- `String colorPlayerA`
- `String namePlayerBStr`
- `String colorOpponenent`

- ---------- Methods ----------
  - `void printNewMessage(Message mess, String sender)` : Affiche le MP de l'adversaire dans le chat
  - `void setProperties(StackPane root)`
  - `void setPlayersNames(String playerA, String playerB)`
  - `void setPlayersColors(String colorOpp)`
  - `GridPane setUpBoard()` : appelé dans `MainClient.switchSceneToGame()`
  - `void setUpPieces(String colorBottom)` : Sets up pieces disposition so that current player has its pieces at the bottom
  - `void placePiece(int row, int col, String pieceName)` : Place une pièce sur un case
  - `void removePiece(int row, int col)` : retire une pièce d'une case
  - `void handleSquareClick(int row, int col)` : gère les cliques du joueur sur l'échéquier
  - `void movePiece(String ordre)` : Reçois ordre du serveur (envoyé aux 2 clients) en réponse à la requête (demande de déplacement d'une pièce) d'un des 2 clients/joueurs
  - `ImageView getPieceAt(int row, int col)` : retourne l' `ImageView` se trouvant à la position [row,col] de `boardPane`
  - `Rectangle getSquareAt(int row, int col)` retourne le `Rectangle` se trouvant à la position [row,col] de `boardPane`
  - `String getColorPlayerA()`