## Specification for Question 2-2, 2-3

### Specifications for User class

- Map<Pair<Integer, Integer>, Integer> bettingIdMap
    - Key: Pair which contains (matchId, betting option). For example, the third betting option of match 10 will correspond to Pair(10, 3). You should contain the contents for the user's submitted bettings (by bet method) whether they're accepted or not.
    - Value: Betting Id of the user's betting assigned during the execution of the server's collectBettings method. Once the Id is assigned, it never changes. Also, betting Id is equivalent to the written order on the bettingBook of the bettings on a specific option.

For example, if the users bet on the same match in the following order (collectBettings traverse order -> user#1 earlier than user#2):

- User#1,2,3 bet on option#1 at time 0. Then the server collected.
- User#1,4,5 bet on option#1 at time 1. Then the server collected.
- User#1 bet on option#2, user#3 bet on option#3, user#5 bet on option#4, user#6 bet on option#1 at time 2. Then the server collected.
- User#1 bet on option#4, user#2 bet on option#2, user#3 bet on option#1, user#5 bet on option#3 at time 3. Then the server collected.

betting Id should be:

- User#1: Id=1 for option#1,2. Id=2 for option#4.
- User#2: Id=2 for option#1,2.
- User#3: Id=3 for option#1. Id=1 for option#3.
- User#4: Id=4 for option#1.
- User#5: Id=5 for option#1. Id=2 for option#3. Id=1 for option#4.
- User#6: Id=6 for option#1.

The content of the user#1's "newBettings.txt" can be:

- (betting Info on Option#1)
- (betting Info on Option#1)
- (betting Info on Option#2)
- (betting Info on Option#4)

or **you can merge** the first and second line since they have the same matchId and bettingOption. It doesn't matter.

The content of the "bettingBook.txt" should be:

- (betting Info of User#1, Option#1)
- (betting Info of User#2, Option#1)
- (betting Info of User#3, Option#1)
- (betting Info of User#4, Option#1)
- (betting Info of User#5, Option#1)
- (betting Info of User#1, Option#2)

        - (betting Info of User#3, Option#3)
        - (betting Info of User#5, Option#4)
        - (betting Info of User#6, Option#1)
        - (betting Info of User#1, Option#4)
        - (betting Info of User#2, Option#2)
        - (betting Info of User#5, Option#3)

- int bet (int matchId, int bettingOption, int coin)
    - User will submit a betting by writing to “newBettings.txt”
    - Bet (coin: coin) to (match: matchId), (option: bettingOption)
    - Update bettingIdMap with (Key: Pair(matchId, bettingOption), Value: -1) if there was no previous successful betting on the same option. Otherwise, you don’t need to update the map.
    - Write the betting to the “newBettings.txt”
    - Decrement the totalCoin by coin
    - Return ErrorCode.SUCCESS if there was no problem, or return corresponding error codes if there are some errors. Possible error codes are
        - ErrorCode.IO_ERROR: IO error occurred
        - ErrorCode.NOT_ENOUGH_COINS: user doesn’t have enough coin to bet
        - ErrorCode.OVER_MAX_BETTING: after this betting, the coins bet on all the options of this match will exceed MAX_COINS_PER_MATCH
        - ErrorCode.NEGATIVE_BETTING: coin is zero or negative integer

  This method will be called at the test code.

- int updateBettingId (int matchId, int bettingOption, int newBettingId)
    - Update the content of bettingIdMap with (Key: Pair(matchId, bettingOption), Value: newBettingId)
    - You can freely choose the **return value** of this method
    - Remember, if there’s previously assigned betting Id, which is positive integer, the content of bettingIdMap shouldn’t be updated, which means new Id shouldn’t be assigned by the server.

  This method will be called at the server’s collectBettings method

**Specifications for Server class**

- int collectBettings()
    - Collect the users’ bettings by traversing the user directories and reading the “newBettings.txt” files of all the users. Traverse order should be in ascending order of userId (e.g. 2020-11111 earlier than 2022-00000, 2022-00000 earlier than 2022-11111).
    - I **recommend** returning ErrorCode.SUCCESS if there was no problem. If any IO error occurred, return ErrorCode.IO_ERROR. However, if you want, you **can change** the return value of this method. (It’s not included in the grading

criteria)
    - This method reads each line of the “newBettings.txt” files of all the users and assigns new betting Id if there was no previous betting on the same option for that user. This calls the user’s `updateBettingId` method. If there was previous successful betting on the same option, you should merge it (increment the coins bet on that option) on the “bettingBook.txt” file. This means, each user can appear on the “bettingBook.txt” file at most (`#bettingOptions`) times.
    - As explained before, betting Id is equivalent to the written order of that betting among the bettings on the same option. If a betting’s Id is 3, there’s exactly two bettings (on the same option) whose line numbers on the “bettingBook.txt” is less than that of betting Id 3.
    - If an error occurs, call `updateBettingId` method and update the content of the user’s bettingIdMap with the error code. Be careful, if there's a previously assigned (positive integer) betting Id, you don’t need to update the map even if the error occurs. Possible errors that can occur are:
        - `ErrorCode.IO_ERROR`: IO error occurred
        - `ErrorCode.INVALID_BETTING`: Invalid betting option
        - `ErrorCode.MATCH_NOT_FOUND`: there’s no match with given `matchId`
        - `ErrorCode.LATE_BETTING`: currentTime is later than the match time.

- `List<Betting> getBettingBook(int matchId)`
    - Literally, you should return a ArrayList of bettings in the betting book of the given `matchId`
    - The order of the list’s contents should be identical to the written order in the file

- `boolean settleMatch(int matchId, int winNumber)`
    - Return true if there’s no problem. If there’s no betting book, return false.
    - For the details of settlement, refer to the document. It’s explained at Question 2-3, and the beginning of Question 2.

## Q&As

- Question 1
    - Start Time is always earlier than or equal to End Time
    - Log's End Time is always earlier than or equal to lecture's End Time
    - You can import IOException
    - No duplicate user Id or lecture Id
    - You can import Pair to any codes
- Question 1-1
    - If there are multiple start & end times, accumulate the connection time of each pair
    - No early entrance
    - All users have a "log0.txt" file
- Question 1-2
    - Total connection time should be union of the log files
    - Absent, late, attended is determined with the same criteria as Question 1-1
    - For early entrance, the date can change.
    - More than 2 log files can exist
    - Users can't have both a log file for excused absence and other normal log files.
- Question 2-2
    - error code -41 is lower than -40
    - Current time is set using the `setCurrentTime` method of server at the test code
- Question 2-3
    - You shouldn't initialize bettingIdMap, bettingBook or match information after settlement