# CS307 Programming Assignment 3

**Akif Işıtan 29354**

This programming assignment involves the implementation of synchronization mechanisms in a simulated shared basketball court with threads representing players and referees.

The Court class has the following methods: **constructor()**, **enter()**, **play()** and **leave()**. The play method is not implemented, it is left to the consumer of the class to implement. The constructor initializes the state variables, semaphores and barrier. The enter and leave methods will be explained in detail in the following section.

### enter() method

```
function enter():
  Grab enter lock
  Grab match status lock

  While match is ongoing:
    Release enter and match status locks
    Grab numWaiting lock
    Increment numWaiting
    Release numWaiting lock
    Wait until match ends
    Grab numWaiting lock
    Decrement numWaiting
    Release numWaiting lock
    Re-grab enter and match status locks

  Release match status lock
  Grab numPlayers lock
  Increment numPlayers
  Release numPlayers lock

  If referee is required and players are sufficient:
    Grab match status lock
    Set current thread as referee and start match
    Print starting match message
    Release match status lock
  Else if no referee is required and players are sufficient:
    Grab match status lock
    Start match
    Print starting match message
    Release match status lock
  Else:
    Print waiting message

  Release enter lock
```

The players first begin by calling the enter method. The player first grabs the enter lock to synchronize the method body, then grabs the match status lock to check if the match is ongoing. If there is a match ongoing, it releases the locks, increments the number of waiting players and waits until the final player of the ongoing game to signal. Once it receives the signal, it decreases the number of waiting players, re-grabs the enter and match

status locks and checks if the match is ongoing, repeating the cycle if it is. It is important to note that this will happen only if there are more than enough threads waiting for a match, meaning that it is not an infinite process. If there is not a match ongoing, the player releases the match status lock, increments the number of players atomically, then checks if a referee is required. If a referee is required and there are already enough players, it grabs the match status lock, assigns itself as the referee and starts the match, then releases the match status lock. Else if a referee is not required and there are enough players, it grabs the match status lock, starts the match and releases the match status lock. Else, which means that there are not enough players, it just prints its line and continues. After these operations, the player releases the enter lock, returning from the enter method.

This method uses binary semaphores as locks to ensure atomicity and correctness on shared resources, and a semaphore initialized to 0 as a sort of waitlist for players that attempt to join the court but cannot because of the ongoing game.

### leave() method

```
function leave():
  Grab match status lock

  If match not ongoing:
    Grab numPlayers lock
    Decrement numPlayers
    Release numPlayers lock
    Release match status lock
    Return

  Release match status lock
  Wait at barrier until every player in the game reaches it.
  If referee is required:
    If thread is referee:
      Print referee leaving message
      Wait at barrier to signal that the message was printed
    Else:
      Wait at barrier until referee leaving message is printed
      Print player leaving message
  Else:
    Print player leaving message

  Grab numPlayers lock
  Decrement numPlayers

  If last player:
    Release numPlayers lock
    Grab match status lock
    Grab numWaiting lock
    Notify waiting players
    Release numWaiting lock
    Stop match
    Release match status lock
  Else:
    Release numPlayers lock
```

After the players return from the play method, they call the leave method. The player first grabs the match status lock and checks if a match is ongoing. If a match is not ongoing by the time the player calls the leave method it, atomically decrements the number of players, releases the match status lock and returns. It just leaves, sort of like it did a warmup by itself without actually playing a match. The reason the match status lock is released at the end of the if block and not the beginning is to prevent a condition where the referee starts the match but the player is already leaving, leading to the barrier blocking indefinitely. If a match is ongoing, it releases the match status lock, and waits on the barrier. It does this to ensure that either the referee - if there is one required - or the starting player prints the starting match line before continuing with the leave operation. Once all the players reach the barrier, which indicates that the statement has been printed, the player continues. It then checks if the referee is required. This is because it needs to wait for the referee to leave before leaving. The same barrier is reused to wait for the referee to print its statement and leave before the other players leave. If a referee is not required, the players just print their statements in whichever order. The player then grabs the num players lock, decrements it by one and checks if it's the last player. If it is not, it releases the num players lock and just returns from the method, its job is done. If the player is the last player, it releases the num players lock, grabs the match status lock and the num waiting lock, then wakes up each player waiting on the waitMatchEnd semaphore. It then releases the num waiting lock, sets the matchOngoing variable to false, releases the match status lock and returns from the method.

This method ensures correctness by using a barrier for referee / starting player print statement synchronization, binary semaphores as locks for shared resources and the previously mentioned 0 initialized semaphore to wake up the players on the waitlist.