Contents

[Context 1](#_Toc26730659)

[Betting process 1](#_Toc26730660)

[Player 2](#_Toc26730661)

[Betting Round 2](#_Toc26730662)

[Game machines 3](#_Toc26730663)

[Gambling authority 3](#_Toc26730664)

[Specific (technical) requirements 4](#_Toc26730665)

[Advanced requirements 5](#_Toc26730666)

[Recommendations 5](#_Toc26730667)

# Context

Casinos are lucrative businesses where a lot of turnover is made. To make they work in a fair and legal way, a gambling authority is involved to oversee the way casino’s make betting possible.

This case describes a part of an application which forces a casino to perform bets in a legal way. It focusses on electronic betting using gaming machines (for instance: slot machines, race machines, online gambling)



Figure 1:a Game with 8 Gaming Machines

## Player

A Player can bet in a casino. They own one (or more) playerCards, which they connect to a GamingMachine. On the GamingMachine they can place a Bet on (the current BettingRound in) a Game. To be able to do this, the card generates a unique betID for that specific bet. A list of all generated betID’s is stored on the card.

The gaming machine checks with the cashier if the player can place the bet or not. Gaming machines are notified at the end of every bettinground by the game, to know who won.

The Gaming machine keeps track of which bet has been made during the bettinground of the game. It also notifies the cashier to update the card of the player if he has won.

## Cashier



Figure 2: A cashier counter

Figure 3: Player Cards

Each casino has one Cashier, which hands out playerCards to players, and the player can put an amount (of money) on the card. This amount is never stored on the card, but the Cashier keeps an administration which card has which amount on it. Obviously, a MoneyAmount can be added to the playercard.

When handing in the card at a Bank teller, all bets made with the card (betID’s) are read from the card and logged by the Cashier. Also, the read betID’s from the card are transferred to the gambling authority through their BetLoggingAuthority interface. The total amount of money credit is physically handed to the player (outside the scope of the application), and the amount stored on the card is changed to zero. The stored betID’s on the card are also removed. This way, the card is anonymous, and can be reused.

The cashier can also always check if a bet can be made using a certain card.

## Game and Betting Round

A casino runs several games. A game in a casino consists of having betting rounds, one after another. A betting round is one round in a game where bets can be submitted, after which the game determines who is the winner. There can only be 1 current betting round per game. A bet on a game is submitted through betting machines (to which Players have connected their gambling card). Max 1 bet per machine can be submitted for the current betting round in a game. When a submitted Bet is added to the Game, that excepted Bet is stored by the Betting Round, and is also sent to the Gambling Authority.

When a new bettinground needs to start, the casino creates a unique BettingRoundID. This bettingRoundID is presented to the Gambling authority, which returns a unique GamblingToken. Both the bettingRoundID and gambling token are used when creating a BettingRound. The betting Round consists therefore of the bettingRoundID (created by the casino), and the GamblingToken created by the gambling authority.

At the end of a betting round: To determine which Bet is the winner, the gambling round asks the gambling authority for a random whole-number value (the randomWinValue). This value is used to determine by the GameRule which bet wins.

A betting round keeps track of all bets made during the round. This information is sent to the betlogging authority when the betting round ends.

## Game machines

The casino has gaming machines with a unique gamingmachineID. On a game machine, a player can place a bet by using his card. Before placing a bet on the current bet round, the machine checks with the bank teller if the card has enough credit to place the bet amount. If so, the current credit of the card is decremented, and a bet is submitted by the betting machine to the game. This way, the current credit of a card is only known by the bank teller, and only the machine knows which bet is made.



Figure 4: Fishing game with 6 gaming machines

## Gambling authority

Using covert people and other systems (both outside of the scope of this assignment), the gambling authority checks if all bets have been logged, and it finds suspicious behaviour by checking times of betting and logging.

To be able to do this, the gambling authority provides two interfaces each casino is forced to use. (see bettingauthorityAPI package)

One is the BetTokenAuthority interface. All calls to the BetTokenAuthority interface are internally timestamped and logged by the gambling authority (code is outside of the scope). As can be seen in the Authority’s API, it can create BetTokens: A BetToken consists of a unique number, and a timestamp when it was created.

It also can provide a true random integer to determine the outcome of each betting round in a game.

The other functionality is the BetLoggingAuthority, which is used for logging all betting activity and player card information. Obviously, all calls to the gambling authority are internally timestamped. (code is outside of the scope).

# Specific (technical) requirements

1. Create tests and code which implement given interfaces.
2. Create Exceptions when called for by the interface.
3. A generalID consists of a unique ID (using the UUID type of java), together with a TimeStamp. Both parts are created during generalID construction. Specific ID types (CardID, BetID, BettingRoundID, GamingMachineID) are derived from generalID. All types of ID need to implement the Comparable interface.
4. All ID objects need to be created by an IDFactory, using the Factory pattern. Create tests for the IDFactory as well.
5. A game is defined by its interface IGame. Enforce injection of required classes by defining an abstract class AbstractGame, implementing IGame, which has the right constructor for this.
6. You can apply the same principle for other classes as well.
7. The application at least should be able to handle synchronous communication.
8. Do NOT create or add test & code for:  
   The whole bettingauthorityAPI package  
   Immutable classes like Bet, BetResult, MoneyAmount  
   Generated code by IntelliJ

# Advanced requirements

Use asynchronous communication for notification of winning bets.

Create Player class which actually perform bets.

Use multiple threads simulating players.

Make the application location aware (time zones)

# Recommendations (strongly!)

* Create a list of requirements based on the description above, and the given interfaces.
* Draw a class diagram of the whole application.
* For each requirement: Define a list of tests for the behaviours. Use the class diagram to find out when direct or indirect in/output is necessary for each test.
* Per test: determine (and justify) first, what kind of test type is needed (f.i. decision points test, path coverage test).
* Use TDD.