## Requisitos

Como punto de partida, se han tomado los requisitos básicos enunciados en el ejercicio propuesto, ampliándolos con nuevos supuestos surgidos a medida que se avanzaba iterativamente en el desarrollo del proyecto.

Con base en la simulación simple por consola del juego que se pedía en el primer ejercicio *("Ejercicio Oca 2014")*, en esta versión para red se han añadido las funcionalidades necesarias para dar soporte a la comunicación entre jugadores.

También se ha realizado el diseño de una interfaz gráfica completa organizada en diferentes áreas para la visualización del tablero con sus fichas, mensajes generados durante el desarrollo de la partida, lista de jugadores activos, posibles errores y área de control, con botones para iniciar partida, lanzar dados y finalizar.

Se ha definido un protocolo al que se deben ajustar las comunicaciones entre los jugadores que intervengan en la partida. El modelo que se ha elegido no se basa en una arquitectura cliente/servidor clásica, con una instancia del programa que centralice y coordine las comunicaciones con jugadores/clientes al uso. En su lugar se ha optado por una arquitectura más descentralizada en la que todos los jugadores actúan como cliente y como servidor en el transcurso de la partida para establecer como resultado conexiones directas punto a punto con el resto de participantes.

Uno de los jugadores, cuya dirección IP y puerto son conocidos por los demás, actúa inicialmente como *anfitrión* o *iniciador* de la partida (servidor); el resto (clientes, en esta primera fase de establecimiento de las conexiones) comunica con el anfitrión para apuntarse a la partida, siendo incluidos en la lista de jugadores activos. Los parámetros con que se lanza programa determinan qué rol se desempeña, en función de que se incluya la dirección IP y el puerto a los que conectarse, o sólo el nombre de jugador.

Será el anfitrión quien dé comienzo a la partida (el botón *"Empezar"* estará habilitado únicamente para este jugador), manteniéndose en tanto a la escucha de las conexiones entrantes de nuevos jugadores que deseen participar, según el protocolo que se describe en detalle más adelante.

De acuerdo con este protocolo, cada jugador que conecte con el anfitrión para apuntarse recibirá la información necesaria para establecer conexión mediante sockets con éste y con el resto de jugadores que se hayan conectado hasta ese instante; será el nuevo jugador quien, a su vez, deba comunicar con éstos últimos (que aún desconocen su existencia) para "presentarse" como *invitado* y establecer las correspondientes conexiones, quedando desde ese momento, y hasta que el anfitrión inicie la partida, a la espera de los nuevos jugadores que deban presentarse (mediante un *ServerSocket*).

El anfitrión esperará una confirmación de cada nuevo jugador antes de iniciar el juego, para asegurarse de que queden establecidas correctamente las conexiones entre todos los jugadores.

Del mismo modo que distinguimos entre anfitrión e invitados en la fase de inicialización de la partida, durante su desarrollo, desde el punto de vista de cada participante, habrá un jugador local, con el que se interviene, y unos jugadores remotos, que actuarán como representantes o proxies del resto de participantes con los que mantiene conexiones activas.

# Diseño

## Package modelo

La modelización básica del dominio se realizó ya para la versión inicial del juego como simulación de partida por consola; a este diseño se han añadido las nuevas funcionalidades descritas para la comunicación en red y la utilización de una interfaz gráfica.

Se han identificado inicialmente las siguientes clases:

*OcaEnRed*:

> Será la clase principal que contiene el método *main()*. Servirá de *controller*, identificándolo, según la definición de este tipo de objeto, como la entidad que representa al sistema en su conjunto, como un objeto raíz, dispositivo especializado o subsistema principal, y que tiene la responsabilidad de gestionar los mensajes de operación de sistema.
>
> En la versión simplificada de la simulación por consola no se modeló una clase *Partida;* se añade en el diseño ampliado para el juego de Oca en Red y contendrá toda la lógica de juego.

*Tablero:*

> Se asocia con el tablero de juego en el que se desarrolla la partida. Tendrá conocimiento de las casillas que lo componen, así como de los jugadores que participan (se ha hecho una simplificación de diseño eliminando una clase *Ficha* que se había identificado inicialmente).
>
> Entre sus responsabilidades está el cálculo de la posición de un jugador (ficha asociada) a partir del resultado de tirar los dados y de su posición de partida.

*Casilla:*

Modela las distintas casillas que componen un tablero cásico del juego de la Oca. Se ha definido una clase base abstracta *"Casilla"* a partir de la que se crea una jerarquía de herencia para la tipificación de las casillas en función de los efectos que provocan en el jugador que cae en ellas.

A este respecto, como es común, se asocian los atributos y comportamientos comunes con la clase base, modelando los comportamientos específicos en los subtipos.

Se hace una primera distinción entre *CasillaNormal, CasillaEspera* y *CasillaDesplazamiento*. Al primer tipo corresponden las casillas sin consecuencias específicas en el juego.

En el segundo grupo incluimos las que conllevan una penalización para el jugador como un determinado número de rondas sin jugar: *CasillaCarcel*, *CasillaPosada* y *CasillaPozo*.

En el tercero están todas las que implican un cambio de posición del jugador: *CasillaDados*, *CasillaLaberinto*, *CasillaMuerte*, *CasillaOca* y *CasillaPuente*.

Se identifican dos casillas especiales, que heredan directamente de la clase abstracta *Casilla*: *CasillaInicio* y *CasillaNormal.*

En la clase base se implementan los campos y métodos comunes a todas las casillas, como el índice en tablero, la casilla a la que se traslada el jugador que llega (la misma para todas excepto las de tipo desplazamiento), una referencia al jugador asociado, una referencia al tablero, y el comportamiento base para las operaciones *entrar()* y *salir()*.

Los subtipos introducen comportamiento específico reemplazando o ampliando polimórficamente este comportamiento base.

*Dado:*

Representa cada uno de los dos dados con que se juega.

Están parametrizados para admitir el número de caras que se especifique en su constructor.

Como característica de diseño, destacar que los dados se asocian a la partida, por lo que serán compartidos por los jugadores.

Siguiendo el principio de separación *"Command/Query",* definimos un método para el lanzamiento, que actualiza el valor del dado, y un método para la recuperación del valor obtenido en la última tirada.

*Jugador:*

Esta clase representa a cada jugador que interviene en la partida.

Guarda los principales campos de identificación y estado relativos al jugador, como su nombre, turno asignado y su situación en la partida.

En la ampliación para juego en red puede pensarse en los objetos jugador como representantes o *proxies* de los jugadores físicos conectados a la misma partida, con campos relativos a la conexión (dirección IP, puerto y socket con sus correspondientes *streams* de entrada y salida). De acuerdo a esta observación, definimos dos subtipos:

*JugadorLocal:*

Representa al jugador con que participamos.

*JugadorRemoto:*

Representa a cada uno de los jugadores conectados.

En la clase base se definen como abstractos los métodos que deberán implementar las comunicaciones con el jugador representado, así como el método más importante de la clase *jugarTurno()*.

Independientemente del tipo de jugador, cualquiera puede desempeñar el rol de *Anfitrión* o de *Invitado*.

Esta clase extiende de *Thread* para que se puedan atender las comunicaciones con el resto de jugadores sin que por ello se bloquee el hilo principal. La lógica para el tratamiento de los mensajes enviados/recibidos a/desde otros jugadores, de acuerdo al protocolo, se implementa en la redefinición del método *run()* que se ejecuta en un hilo independiente.

## Package vista

Todas las clases necesarias para la visualización del juego mediante una interfaz gráfica de usuario (GUI) se incluyen en este paquete.

Se ha procurado respetar la separación entre las capas de presentación y la capa de negocio, de acuerdo al patrón MVC *(Model/View/Controller)*.

La principal clase es *Interfaz*, que actúa como "fachada" (patrón *façade*) para el resto de clases.

Identificamos las clases *TableroOcaGUI*, *DadoGUI* y *FichaGUI* para la representación de los correspondientes elementos.

Se pretende que el acoplamiento con las clases de negocio se mantenga en un mínimo, por lo que sólo representan gráficamente el estado de los objetos de negocio, sin incluir lógica de juego.

Idealmente, la relación de estas clases con el resto se debe basar en el patrón *Observer*.

## Package util

Las características de las casillas que componen el tablero se definen en un fichero XML (*tablero.xml*). Para cada casilla se definen los siguientes atributos y elementos:

*indice:* Atributo del elemento *Casilla.* Define el índice de la casilla en el tablero

*Tipo:* tipo de casilla:

- *CASILLA_INICIO = 1;*
- *CASILLA_NORMAL = 2;*
- *CASILLA_OCA = 3;*
- *CASILLA_PUENTE = 4;*
- *CASILLA_POSADA = 5;*
- *CASILLA_DADOS = 6;*
- *CASILLA_POZO = 7;*
- *CASILLA_LABERINTO = 8;*
- *CASILLA_CARCEL = 9;*
- *CASILLA_MUERTE = 10;*
- *CASILLA_FIN = 11;*

*IndiceDestino:* índice al que se redirige el jugador, según tipo de casilla.

*RondasSinJugar*: Penalización. Rondas sin jugar, según tipo de casilla.

*RepetirTurno*: Bonificación. El jugador vuelve a tirar, según tipo de casilla.

*CoordenadasGUI*: Coordenadas gráficas.

Se define un tipo *CasillaInfo* para recoger la parametrización de cada casilla en la lectura del XML.

Para procesar el fichero XML se define la clase *CasillaInfoReader*, que se basa en las librerías STAX *(Java Structured API for XML)*.

Determinados atributos son de interés para la lógica de aplicación, mientras que otros se necesitan sólo en la capa de GUI.

## Protocolo de comunicación

Se define un protocolo que regirá la comunicación entre los jugadores; los tipos de mensajes que pueden intercambiarse se enumeran en la clase *Protocolo*.

Un mensaje intercambiado entre dos jugadores puede ser de uno de los siguientes tipos:

- *OK*: Para la confirmación positiva de mensaje recibido
- *KO*: Para la confirmación negativa de mensaje recibido
- *NUEVO*: Mensaje que envía al anfitrión el jugador que desea participar
- *INVITADO*: Mensaje que envía un nuevo jugador al resto de los jugadores conectados
- *INICIO:* Mensaje de comienzo de partida
- *TIRADA*: Mensaje para comunicar el resultado de la tirada de dados
- *PRESENTADO*: Mensaje para confirmar que el nuevo jugador ha terminado de presentarse al resto de participantes

Los dos primeros se proporcionan como confirmación afirmativa y negativa, respectivamente, a otros mensajes del protocolo.

Al inicio de la partida, el jugador que tiene el rol de *anfitrión* o *iniciador* pasa a escuchar en un *ServerSocket* todas las comunicaciones entrantes del resto de participantes que quieran apuntarse a la partida.

El tipo de jugador asociado al programa se determina mediante los parámetros con que se lanza: si no se especifican una dirección IP y puerto remoto (los del anfitrión) se asume que el jugador es el iniciador; en caso contrario, se intenta la conexión con el anfitrión en la dirección IP y puerto proporcionados para inscribirse en la partida.

En esta primera fase de inicio, el anfitrión espera un mensaje de tipo *NUEVO* por parte de cualquier comunicación entrante como solicitud de nuevo jugador. El mensaje de tipo *NUEVO* debe ir seguido del nombre o identificador del jugador, dirección IP y puerto remoto de conexión.

Como respuesta, por el nuevo socket creado, el anfitrión envía al jugador su propio nombre, dirección IP, puerto local y turno para que pueda éste configurar su conexión.

A continuación se envían los mismos datos de todos los jugadores que estuviesen conectados hasta el momento.

El jugador debe ponerse en contacto con cada uno de los jugadores conectados para establecer el socket correspondiente. Cada jugador estará representado por un objeto *JugadorLocal* en el que se encapsula el socket y los *streams* para la comunicación.

Tras el establecimiento de las comunicaciones con el resto de jugadores, envía una confirmación al anfitrión para que finalice su inclusión en la partida (de modo que no se inicie el juego si aún no han terminado de establecerse todas las conexiones). Es un mensaje de tipo *PRESENTADO*.

El anfitrión seguirá actuando como servidor durante la fase de iniciación de partida, aceptando las conexiones entrantes que se ajusten al protocolo por medio del *ServerSocket* creado en la clase *Partida*, hasta el momento en que decida iniciar juego pulsando el botón correspondiente en la interfaz.

Durante esta misma etapa de inicialización, los jugadores que se van incorporando a la partida pasan a actuar como servidores, esperando conexiones entrantes de los jugadores que se hayan apuntado a posteriori para establecer la comunicación mediante el correspondiente socket.

Cuando el iniciador decide comenzar la partida envía a todos los participantes (a través de los objetos *JugadorRemoto* creados hasta ese momento) un mensaje *INICIO*, con lo que se pasa a la siguiente fase de juego.

En esta fase se juegan sucesivas *rondas*, en las que cada jugador tiene su *turno,* hasta que finaliza la partida.

En cada ronda el jugador que tiene el turno es responsable de jugar; el comportamiento polimórfico de los distintos tipos de jugadores (remotos y locales) es el que determina si el jugador lanza realmente los dados y actualiza su posición (local) o si se espera a recibir el mensaje de *TIRADA* con el resultado de los dados lanzados por un jugador remoto.

Este tipo de mensaje va seguido de los valores de los dos dados y lo envía el jugador local a todos los jugadores de la partida, para que puedan simular el lanzamiento de los dados y el movimiento de la ficha en su propia interfaz gráfica.

## Interfaz Gráfica de Usuario

La interfaz gráfica de usuario está dividida en tres áreas principales:


- *Tablero de juego* con las fichas de los participantes. Se ha utilizado la imagen proporcionada en el enunciado del ejercicio.
- *Área de información*, con un área de mensajes de juego, una lista de jugadores activos y un área de mensajes de error.
- *Área de control*, con un botón para iniciar la partida (habilitado únicamente para el anfitrión), un botón para lanzar los dados (habilitado únicamente para el jugador que tiene el turno), las imágenes de los dados (animadas) y un botón para abandonar la partida.

Se definen las siguientes clases:

- *Interfaz*: Clase principal que sirve de fachada para el resto del sistema. Crea y mantiene las relaciones con el resto de clases de la interfaz gráfica.
- *TableroOcaGUI*: muestra la imagen del tablero, mantiene la lista de fichas y tiene la responsabilidad de simular su movimiento a partir de la posición de inicio y la casilla de destino. Se deriva de la clase *Canvas*, redefiniendo su método *paint()* para que pueda dibujarse en pantalla. No se borra el contenido anterior para evitar un efecto de parpadeo (*flickering*); muestra en primer lugar la imagen del tablero y a continuación la fichas de cada jugador (formato png con definición de transparencia).
- *FichaGUI*: representa una ficha. Mantiene la imagen de la ficha y los principales atributos para su identificación, como turno, índice de casilla donde se ubica y coordenadas gráficas.
- *DadoGUI*: representa un dado. Mantiene las seis imágenes correspondientes a las seis caras de un dado; es responsable de la animación de tirada, así como de mantener el valor obtenido.

## Ampliación

Por restricciones en el alcance del proyecto a consecuencia del tiempo (muy) limitado disponible (tiempos libres a lo largo de dos semanas para el desarrollo de la versión consola y de su ampliación como juego en red), no ha sido posible profundizar en una serie de aspectos que pueden plantearse como mejoras, algunas de índole meramente estética y otras, más importantes, para mejorar la estabilidad frente a posibles fallos en las conexiones.

Aún cuando se comprueba la conexión con cada jugador de forma periódica para evitar el bloqueo de la partida, detectando los jugadores inactivos para su eliminación, la aplicación debería ser mucho más robusta a fallos de comunicaciones, incorporando en el protocolo reintentos de conexión o reincorporación de jugadores que hayan podido perder la comunicación temporalmente.

Como añadido interesante, y aprovechando las comunicaciones punto a punto que se establecen entre jugadores, podría ampliarse el protocolo para dotar a la aplicación de la posibilidad de intercambiar comentarios (en privado) entre los jugadores que lo deseen, o públicamente con todos a la vez.

## Ejecución

Como ya se ha comentado en la sección correspondiente, existen dos modalidades de jugador en la aplicación: anfitrión e invitado.

El primero inicializa el juego y queda a la espera de jugadores que desean conectar en una dirección IP y un puerto conocidos por todos (para las pruebas se utiliza la dirección de host local 127.0.0.1 y el puerto por defecto que se indica en el enunciado del ejercicio 8888). El anfitrión da comienzo a la partida pulsando el botón "Empezar" de la interfaz gráfica.

El invitado se dirige en primer lugar al anfitrión para iniciar conexión y se comunica con los jugadores que han sido dados de alta hasta ese instante para establecer nuevas conexiones. Queda a la espera de solicitudes de conexión de jugadores invitados a posteriori hasta que el anfitrión decide que comience la partida.

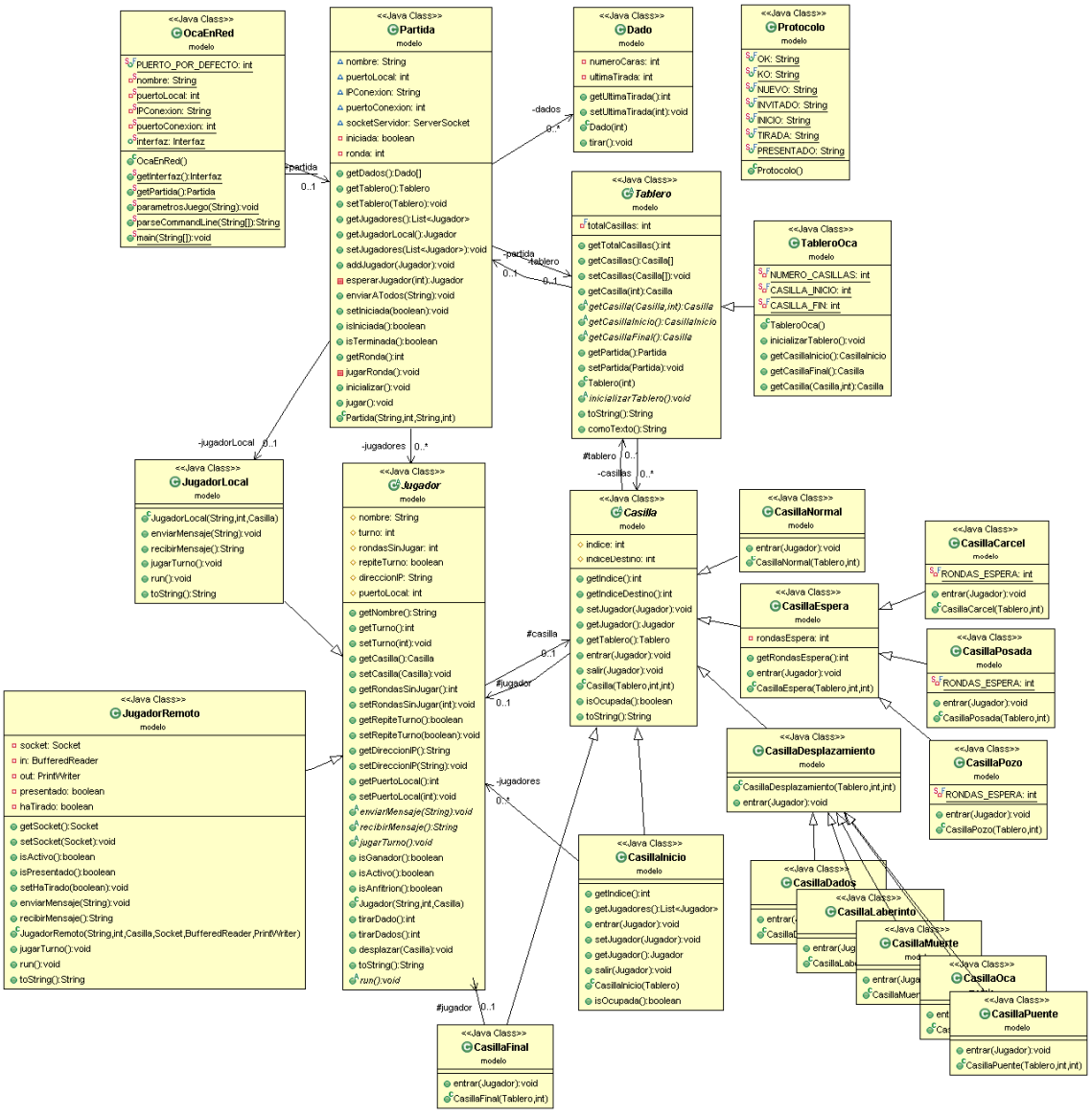
El rol desempeñado se determina por la forma de ejecutar el programa, con unos parámetros u otros:

```
OcaEnRed [ -ip dir_ip [ -r puerto ] ] [ -l puerto ] jugador

     jugador : Nuestra identificación en la partida

         -ip : IP del anfitrión ( no se informa si somos nosotros )

          -r : Puerto de conexion con el anfitrión

          -l : Puerto propio de escucha
```

Para iniciar el juego como anfitrión no se debe especificar una dirección IP ni un puerto remoto de conexión; si se desea jugar como invitado, se deberán informar ambos parámetros.

# UML. Diagrama de Clases.

## Package modelo.



# UML. Diagrama de Clases.

## Package vista.