Universidad ORT Uruguay

Facultad de Ingeniería Escuela de Tecnología

OBLIGATORIO PROGRAMACIÓN 2

Gustavo Brañas – 282436

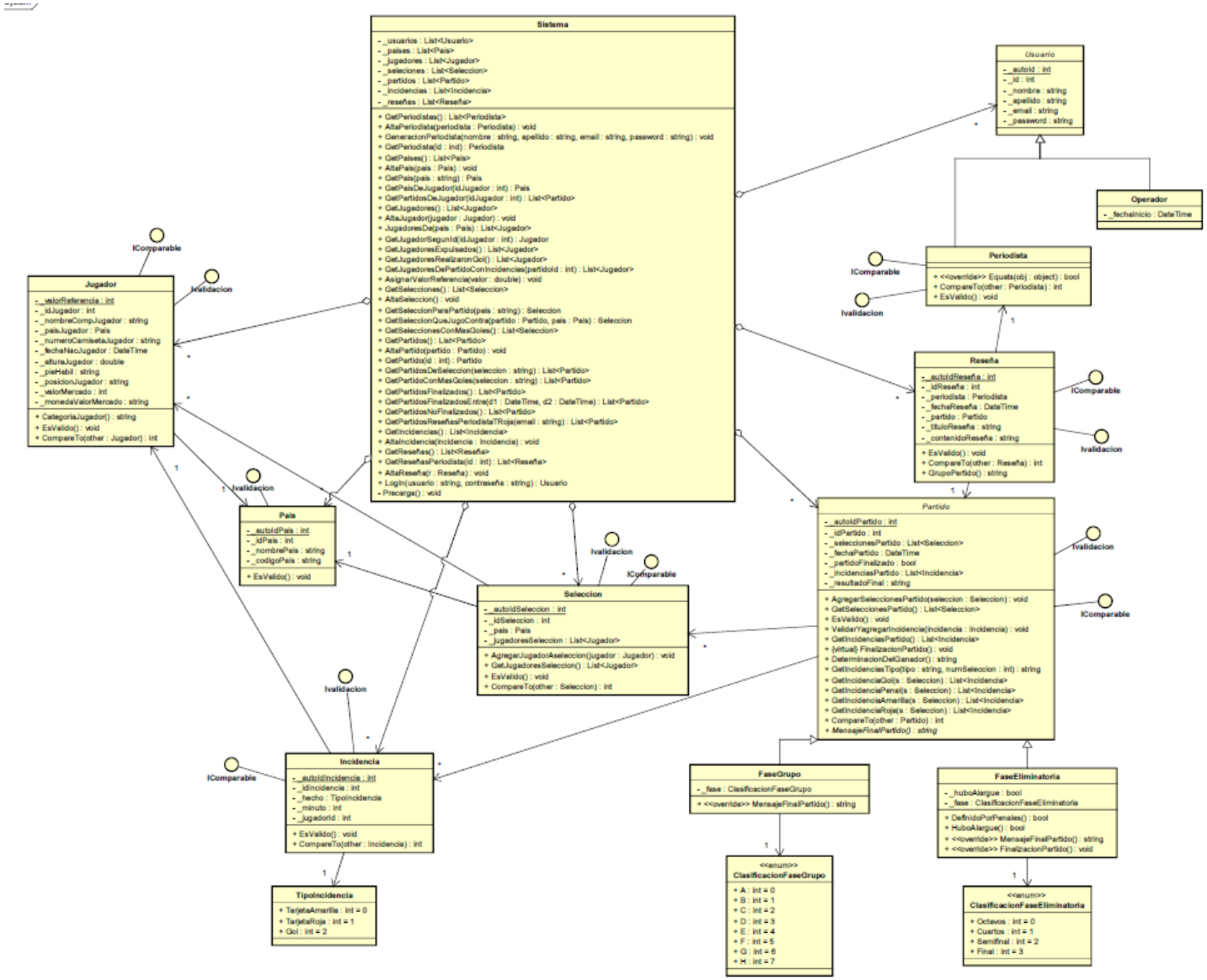
Grupo M2A Remoto

Docente: Joaquín Rodriguez

Analista en Tecnologías de la Información

Índice

1.0 Diagrama de clases



2.0 Deploy en Somee

http://Programacion2OB.somee.com

3.0Diagrama de casos de uso

Caso 1: Funcionalidades de usuario sin identificar

Caso 1A: Log In	
Fecha: 15/11/2022	
Descripción	Acceso al sistema mediante validación de credenciales
Actor	Usuario sin registro
Precondición	El usuario debe de haberse registrado como periodista o encontrarse precargado en el sistema (Usuario Operador o Periodista)
Flujo normal	
1) Se ingresa a Home y se lo recibe en la página de Log In. 2) Se solicita al usuario que ingrese mail y contraseña. 3) Si los datos son aceptados (ningún campo fue ingresado vacío y existe coincidencia en las credenciales -usuario y contraseña-) el usuario es redirigido a la página Home según su categoría (se da acceso a funcionalidades de Operador o de Periodista).	
Flujo alternativo	
3A – a) El usuario ingresa un campo vacío. b) Se despliega un mensaje que indica, según el campo, “Complete campo Usuario” o “Complete campo Contraseña”	
3B – a) El usuario ingresa un mail y/o contraseña que no machea con los datos registrados en el sistema. b) Se despliega un mensaje que indica “Error de ingreso. Corrobore usuario y contraseña”.	
Flujo excepcional	
No aplica	

Postcondiciones
Se da acceso a funcionalidades según se trate de usuario Operador o Periodista

Caso 2A: Acceso al enlace Selecciones	
Fecha: 15/11/2022	
Descripción	El usuario sin identificar clickea sobre el enlace de selecciones
Actor	Usuario sin registro
Precondición	No aplica
Flujo normal	
1) Se clickea sobre el link y se despliega una pantalla donde se muestra el listado de selecciones ordenadas alfabéticamente ascendentes con sus respectivos jugadores.	
Flujo alternativo	
No aplica	
Flujo excepcional	
No aplica	
Postcondiciones	
No aplica	

Caso 3A: Registro de periodistas	
Fecha: 15/11/2022	
Descripción	Esta funcionalidad permite a un periodista registrarse en la aplicación.
Actor	Usuario sin registro
Precondición	No debe de existir el mail ingresado en el sistema
Flujo normal	

1) Se le solicita al usuario que ingrese Nombre, Apellido, Mail y Contraseña. Se le recuerda al usuario que la contraseña debe de contener al menos 8 caracteres.
2) El usuario completa todos los datos, acepta Términos y Condiciones y además cumple con los requisitos (mail una sola arroba que no debe de encontrarse al comienzo o al final del string y que la longitud de la contraseña debe de ser 8 caracteres).
3) Se despliega un mensaje en color verde que indica que su registro se realizó correctamente.

Flujo alternativo

2A -

a) El usuario ingresa un campo vacío.
b) Se le avisa al usuario que (según sea el caso) debe de ingresar Nombre, Apellido, Email o Contraseña, con el siguiente mensaje en color rojo “Complete campo Nombre / Apellido / Email / Contraseña”.

2B -

a) El usuario ingresa un mail ya registrado.
b) Se le avisa que el valor ingresado no es aceptado, desplegando un mensaje en color rojo que indica “El periodista ya existe en la base de datos”.

2C -

a) El usuario ingresa un mail incorrecto, con una @ al comienzo o al final.
b) Se le avisa que “El mail ingresado no es válido: no posee o posee más de 1 sola @” o “El mail comienza o termina con '@'” desplegando este mensaje en color rojo.

2D -

c) El usuario no acepta Términos y Condiciones.
d) Se le avisa que “Necesita confirmar que Acepta Términos y Condiciones” desplegando este mensaje en color rojo.

Flujo excepcional

No aplica.

Postcondiciones

Se registran los datos del Periodista en la base de datos.

Caso 2: Funcionalidades de usuario Periodista

Caso 2A: Visualización del Home	
Fecha: 15/11/2022	
Descripción	Página de bienvenida al Periodista donde se muestra el nombre y apellido del Periodista.
Actor	Usuario Periodista
Precondición	Periodista loggeado.
Flujo normal	
1) Página de Bienvenida.	
Flujo alternativo	
No aplica	
Flujo excepcional	
No aplica.	
Postcondiciones	
El usuario registrado podrá seleccionar para visualizar Partidos finalizados, realizar Reseñas de estos y a su vez visualizar la redacción de sus propias reseñas.	

Caso 2B: Visualización de Partidos finalizados	
Fecha: 15/11/2022	
Descripción	Se muestra en una grilla los partidos finalizados, incluyendo la fecha del partido, las selecciones que jugaron y el ganador. Se muestra a su vez un enlace para realizar una reseña de un determinado partido.
Actor	Usuario Periodista
Precondición	Periodista loggeado. Existen partidos finalizados.
Flujo normal	
1) Se realiza click sobre el enlace Partidos finalizados Redacción de reseñas.	

Flujo alternativo
No aplica
Flujo excepcional
No aplica.
Postcondiciones
El Periodista puede realizar una reseña de un partido finalizado.

Caso 2C: Realización de reseñas de un partido	
Fecha: 15/11/2022	
Descripción	Se muestra un resumen del partido a reseñar y dos campos de texto para ingresar el título y el texto de la reseña.
Actor	Usuario Periodista
Precondición	Se accede a esta funcionalidad desde la página Partidos finalizados.
Flujo normal	
1) Se despliega la pantalla y el Periodista registrado deberá introducir el título de la reseña y el texto de esta. 2) Si se completan todos los campos, la reseña se ingresa al sistema comunicando al usuario mediante un mensaje en color verde que índica “Se realizó correctamente la reseña”.	
Flujo alternativo	
2A - a) El usuario ingresa algún campo vacío. b) Se le avisa al usuario mediante un mensaje en color rojo que “Existen campos vacíos. Por favor complételos para realizar la reseña”.	
Flujo excepcional	
No aplica.	
Postcondiciones	

La reseña se ingresa al sistema.

<u>Caso 2D:</u> Visualización de reseñas realizadas	
Fecha: 15/11/2022	
Descripción	Se muestran las reseñas realizadas por el periodista de aquellos partidos que se encuentran finalizados.
Actor	Usuario Periodista
Precondición	Periodista loggeado. Existen partidos finalizados.
Flujo normal	
1) Se despliega en pantalla el título de la reseña, las selecciones que participaron del partido y el contenido de esta.	
Flujo alternativo	
No aplica.	
Flujo excepcional	
No aplica.	
Postcondiciones	
No aplica.	

Caso 2C: Log out	
Fecha: 15/11/2022	
Descripción	El usuario cierra sesión.
Actor	Usuario Periodista
Precondición	Periodista loggeado.
Flujo normal	
1) Se clickea sobre el icono de Log out. 2) Se consulta al Periodista si desea cerrar la sesión. Si este acepta, la sesión se cierra y el usuario es redirigido al Home de Usuario no identificado.	
Flujo alternativo	
2A - a) El usuario cancela el Log out. b) La sesión no es cerrada.	
Flujo excepcional	
No aplica.	
Postcondiciones	
El Periodista cierra sesión en la aplicación.	

Caso 3: Funcionalidades de usuario Operador

Caso 3A: Visualización del Home	
Fecha: 15/11/2022	
Descripción	Página de bienvenida al Operador donde se muestra el nombre y apellido del Operador.
Actor	Usuario Operador
Precondición	Operador loggeado.
Flujo normal	
1) Página de Bienvenida.	
Flujo alternativo	
No aplica	
Flujo excepcional	
No aplica.	
Postcondiciones	
El usuario registrado podrá seleccionar para visualizar Selecciones participantes, Partidos finalizados, Finalizar partidos, Filtrar partidos, visualizar Periodistas activos en la aplicación y visualizar Estadísticas.	

Caso 3B: Acceso al enlace Selecciones	
Fecha: 15/11/2022	
Descripción	El usuario Operador clickea sobre el enlace de selecciones
Actor	Usuario Operador
Precondición	Operador loggeado.
Flujo normal	
1) Se clickea sobre el link y se despliega una pantalla donde se muestra el listado de selecciones ordenadas alfabéticamente ascendentes con sus respectivos jugadores.	
Flujo alternativo	

No aplica
Flujo excepcional
No aplica
Postcondiciones
No aplica

<u>Caso 3C:</u> Finalizar un partido	
Fecha: 15/11/2022	
Descripción	Se le permite al operador finalizar un partido
Actor	Usuario Operador
Precondición	Operador loggeado. Existen partidos sin finalizar.
Flujo normal	
1) Se muestra una lista de partidos no finalizados (si existen) de lo contrario se indica “No existen partidos sin finalizar”. Se detalla Fecha del partido y las selecciones involucradas. 2) Mediante click en un enlace, ubicado en cada línea de partido a la derecha, se le permite finalizar el partido. 3) Se redirige al usuario a una nueva vista donde se muestran los detalles del partido que ha sido finalizado: fase de partido, fecha, selecciones involucradas, goles, tarjetas amarillas y rojas junto con la identificación de la selección ganadora. En caso de ser Fase Octavos de final, se detallan minutos y jugadores que estuvieron implicados en las incidencias.	
Flujo alternativo	
No aplica	
Flujo excepcional	
No aplica.	
Postcondiciones	
El partido se registra en el sistema como Finalizado.	

Caso 3D: Partidos finalizados	
Fecha: 15/11/2022	
Descripción	Se le permite al operador visualizar los partidos finalizados
Actor	Usuario Operador
Precondición	Operador loggeado. Existen partidos finalizados.
Flujo normal	
1) Se muestra una lista de partidos finalizados (si existen) de lo contrario se indica "No existen partidos finalizados". Se detalla Fecha del partido, las selecciones involucradas y el ganador.	
Flujo alternativo	
No aplica	
Flujo excepcional	
No aplica.	
Postcondiciones	
No aplica	

Caso 3E: Filtrar partidos	
Fecha: 15/11/2022	
Descripción	Se le permite al operador filtrar partidos finalizados según un periodo de tiempo establecido por el.
Actor	Usuario Operador
Precondición	Operador loggeado. Existen partidos finalizados.
Flujo normal	
1) Se muestran dos inputs donde se debe seleccionar una fecha inicial y una fecha final para realizar el filtro. 2) El operador selecciona dos fechas, la primera anterior o igual a la segunda.	

3) Se muestran los partidos jugados para ese período de tiempo siempre y cuando estos estén finalizados. De no encontrarse partidos, se muestra un mensaje en rojo que indica “No existen partidos finalizados en el período establecido”.
4) En la vista se detalla fecha del partido, selecciones involucradas, goles, tarjetas amarillas y rojas junto con la identificación de la selección ganadora.

Flujo alternativo

2A -

a) El usuario selecciona dos fechas: la inicial posterior a la segunda.
b) Se despliega un mensaje en color rojo que indica “La segunda fecha debe de ser mayor o igual a la primera. Verifique”.

Flujo excepcional

No aplica.

Postcondiciones

No aplica

Caso 3F: Visualización de periodistas activos	
Fecha: 15/11/2022	
Descripción	Se le permite al operador visualizar los periodistas activos en el sistema.
Actor	Usuario Operador
Precondición	Operador loggeado. Existen periodistas activos.

Flujo normal

1) Se muestran los detalles de los periodistas activos registrados en el sistema. Se detalla apellido, nombre, identificador de periodista y email.
2) Cada línea de periodista cuenta con un link para acceder a las reseñas de partidos finalizados realizadas por dicho periodista.

Flujo alternativo

No aplica.

Flujo excepcional
No aplica.
Postcondiciones
Permite al operador acceder a las reseñas realizadas por un determinado periodista.

<u>Caso 3G:</u> Visualización de reseñas de un determinado periodista.	
Fecha: 15/11/2022	
Descripción	Se le permite al operador visualizar las reseñas redactadas de un determinado periodista.
Actor	Usuario Operador
Precondición	Operador loggeado. Existen reseñas de partidos finalizados y de periodistas activos en el sistema.
Flujo normal	
1) Se muestran los detalles de las reseñas realizadas por un determinado periodista. Se indica fecha de la reseña, fecha del partido, grupo, selecciones involucradas y texto de la reseña.	
Flujo alternativo	
No aplica.	
Flujo excepcional	
No aplica.	
Postcondiciones	
No aplica	

Caso 3H: Visualización de estadísticas.	
Fecha: 15/11/2022	
Descripción	Se le permite al operador visualizar las estadísticas del sistema.
Actor	Usuario Operador
Precondición	Operador loggeado. Existen partidos finalizados y reseñas sobre partidos donde existió al menos una expulsión.
Flujo normal	
1) Se muestra la selección con más goles considerando solo goles en tiempo de juego y partidos finalizados. 2) Esta vista permite buscar partidos donde se hayan redactado reseñas de aquellos con al menos una expulsión. La búsqueda se realiza por mail de periodista. 3) De existir, se detallan lo partidos reseñados, incluyendo fecha del partido, selecciones involucradas, goles, tarjetas amarillas y rojas junto con la identificación de la selección ganadora.	
Flujo alternativo	
3A - a) En caso de no existir reseñas con dichas características, se indica “No existen reseñas para este periodista”.	
Flujo excepcional	
No aplica.	
Postcondiciones	
No aplica	

Caso 3I: Log out	
Fecha: 15/11/2022	
Descripción	El usuario cierra sesión.
Actor	Usuario Operador
Precondición	Operador loggeado.
Flujo normal	
3) Se clickea sobre el icono de Log out. 4) Se consulta al Operador si desea cerrar la sesión. Si este acepta, la sesión se cierra y el usuario es redirigido al Home de Usuario no identificado.	
Flujo alternativo	
2A - c) El usuario cancela el Log out. d) La sesión no es cerrada.	
Flujo excepcional	
No aplica.	
Postcondiciones	
El Operador cierra sesión en la aplicación.	

4.0 Código fuente de clases trabajadas

4.1 Clase Partido

```
public abstract class Partido : IValidacion, IComparable<Partido>
    {
        private static int AutoIdPartido { get; set; } = 1;
        public int IdPartido { get; set; }
        private List<Seleccion> SeleccionesPartido { get; set; }
        public DateTime FechaPartido { get; set; }
        public bool PartidoFinalizado { get; set; }
        private List<Incidencia> IncidenciasPartido { get; set; }
        public string ResultadoFinal { get; set; }


        //CONSTRUCTOR
        protected Partido(DateTime fechaPartido)
        {
            IdPartido = AutoIdPartido;
            AutoIdPartido++;
            SeleccionesPartido = new List<Seleccion>();
            IncidenciasPartido = new List<Incidencia>();
            ResultadoFinal = "Pendiente";
            PartidoFinalizado = false;
            FechaPartido = fechaPartido;
        }

        //METODOS

        //  --> agregar seleccion al partido
        public void AgregarSeleccionesPartido(Seleccion seleccion, Seleccion seleccion2)
        {
            SeleccionesPartido.Add(seleccion);
            SeleccionesPartido.Add(seleccion2);
        }


        //  --> get selecciones del partido
        public List<Seleccion> GetSeleccionesPartido()
        //no virtual no abstract para no modificar
        {
            return SeleccionesPartido;
        }


        //  --> validar partido
        public void EsValido()
        {
            if (SeleccionesPartido.Count != 2 || SeleccionesPartido[0].Pais ==
SeleccionesPartido[1].Pais || FechaPartido > new DateTime(2022, 12, 18) || FechaPartido < new
DateTime(2022, 11, 20))
            {
                throw new Exception("El partido ingresado no es valido");
            }
        }


        //  --> verificar que el jugador exista en la seleccion y agregar la incidencia a la
lista de incidencias del partido
        public void ValidarYagregarIncidencia(Incidencia incidencia)
        {
            if (!PartidoFinalizado)
            {
```

```
                foreach (Seleccion item in GetSeleccionesPartido())
                {
                    foreach (Jugador item2 in item.GetJugadoresSeleccion())
                    {
                        if (item2.IdJugador == incidencia.JugadorId)
                        {
                            IncidenciasPartido.Add(incidencia);
                            break;
                        }
                    }
                }

            }
        }


        //  --> get Incidencias
        public List<Incidencia> GetIncidenciasPartido()
        {
            //no virtual ni abstract para no modificar
            IncidenciasPartido.Sort();
            return IncidenciasPartido;
        }


        //  --> finalizacion partido
        public virtual void FinalizacionPartido()
        {
            PartidoFinalizado = true;
            ResultadoFinal = MensajeFinalPartido();
        }


        //  --> Determinacion del ganador
        protected string DeterminacionDelGanador()
        {
            string ganador = "El partido no ha finalizado";

            if (PartidoFinalizado)
            {
                string pais1 = SeleccionesPartido[0].Pais.CodigoPais;
                int golesSeleccion1 = 0;
                int golesSeleccion2 = 0;

                foreach (Incidencia incidencia in GetIncidenciasPartido())
                {
                    foreach (Seleccion seleccion in GetSeleccionesPartido())
                    {
                        foreach (Jugador jugador in seleccion.GetJugadoresSeleccion())
                        {
                            if (incidencia.Hecho == TipoIncidencia.Gol &&
incidencia.JugadorId == jugador.IdJugador && seleccion.Pais.CodigoPais == pais1)
                            {
                                golesSeleccion1 += 1;
                            }
                            else if (incidencia.Hecho == TipoIncidencia.Gol &&
incidencia.JugadorId == jugador.IdJugador && seleccion.Pais.CodigoPais != pais1)
                            {
                                golesSeleccion2 += 1;
                            }
                        }
                    }
                }

                if (golesSeleccion1 > golesSeleccion2) { ganador =
SeleccionesPartido[0].Pais.NombrePais; }
                else if (golesSeleccion1 < golesSeleccion2) { ganador =
SeleccionesPartido[1].Pais.NombrePais; }
```

```
                else { ganador = "Empate"; }
            }
            return ganador;
        }


        //  --> Mensaje final del partido
        public abstract string MensajeFinalPartido();


        //  --> comparacion de partidos ordenados por fecha
        public int CompareTo([AllowNull] Partido other)
        {
            if (FechaPartido.CompareTo(other.FechaPartido) > 0)
            {
                return 1;
            }
            else if (FechaPartido.CompareTo(other.FechaPartido) < 0)
            {
                return -1;
            }
            else
            {
                return 0;
            }
        }


        //  --> Cantidad de Incidencias tipo (roja, amarilla, gol) y seleccion
        public string GetIncidenciasTipo(string tipo, int numSeleccion)
            //numero de seleccion es la posicion 0 y 1
        {
            int count = 0;
            int countPenales = 0;
            Seleccion s = SeleccionesPartido[numSeleccion];

            if (PartidoFinalizado && tipo != "Gol")
            {
                foreach (Incidencia i in IncidenciasPartido)
                {
                    foreach(Jugador j in s.GetJugadoresSeleccion())
                    if (i.JugadorId == j.IdJugador && i.Hecho.ToString().Equals(tipo))
                    {
                        count++;
                    }
                }
            }
            else if (PartidoFinalizado && tipo == "Gol")
            {
                foreach (Incidencia i in IncidenciasPartido)
                {
                    foreach (Jugador j in s.GetJugadoresSeleccion())
                        if (i.JugadorId == j.IdJugador && i.Hecho.ToString().Equals(tipo) &&
!i.Minuto.Equals(-1))
                        {
                            count++;
                        }else if(i.JugadorId == j.IdJugador &&
i.Hecho.ToString().Equals(tipo) && i.Minuto.Equals(-1))
                        {
                            countPenales++;
                        }
                }
                if (countPenales != 0)
                {
                    return count.ToString() + "  (" + countPenales.ToString() + ")";
                }
            }
            return count.ToString();
```

```
}

public List<Incidencia> GetIncidenciaGol(Seleccion s)
{
    List<Incidencia> _ret = new List<Incidencia>();

    foreach(Incidencia i in GetIncidenciasPartido())
    {
        foreach(Jugador j in s.GetJugadoresSeleccion())
        {
            if (i.Hecho.ToString().Equals("Gol") && !i.Minuto.Equals(-1)
                && j.IdJugador.Equals(i.JugadorId))
            {
                _ret.Add(i);
            }
        }
    }
    return _ret;
}

public List<Incidencia> GetIncidenciaPenal(Seleccion s)
{
    List<Incidencia> _ret = new List<Incidencia>();

    foreach (Incidencia i in GetIncidenciasPartido())
    {
        foreach (Jugador j in s.GetJugadoresSeleccion())
        {
            if (i.Hecho.ToString().Equals("Gol") && i.Minuto.Equals(-1)
                 && j.IdJugador.Equals(i.JugadorId))
            {
                _ret.Add(i);
            }
        }
    }
    return _ret;
}

public List<Incidencia> GetIncidenciaAmarilla(Seleccion s)
{
    List<Incidencia> _ret = new List<Incidencia>();

    foreach (Incidencia i in GetIncidenciasPartido())
    {
        foreach (Jugador j in s.GetJugadoresSeleccion())
        {
            if (i.Hecho.ToString().Equals("TarjetaAmarilla")
                && j.IdJugador.Equals(i.JugadorId))
            {
                _ret.Add(i);
            }
        }
    }
    return _ret;
}

public List<Incidencia> GetIncidenciaRoja(Seleccion s)
{
    List<Incidencia> _ret = new List<Incidencia>();

    foreach (Incidencia i in GetIncidenciasPartido())
    {
        foreach (Jugador j in s.GetJugadoresSeleccion())
        {
            if (i.Hecho.ToString().Equals("TarjetaRoja")
                 && j.IdJugador.Equals(i.JugadorId))
            {
                _ret.Add(i);
```

```
                    }
                }
            }
            return _ret;
        }
    }
```

4.2 Clase FaseGrupo

```
    public class FaseGrupo : Partido
    {
        //PROPERTIES
        public ClasificacionFaseGrupo Fase { get; set; }

        //CONSTRUCTOR
        public FaseGrupo (DateTime fechaPartido, ClasificacionFaseGrupo fase) :
base(fechaPartido)
        {
            Fase = fase;
        }

        //METODO

        //  --> mensaje de ganador
        public override string MensajeFinalPartido()
        {
            string mensaje = "Empate";
            if (DeterminacionDelGanador() != "Empate")
                //metodo determinacion del ganador ya implica que el partido este finalizado
            {
                mensaje = $"Ganador: {DeterminacionDelGanador()}";
            }
            return mensaje;
        }
    }
```

4.3 Clase FaseEliminatoria

```
public class FaseEliminatoria : Partido
    {
        //PROPERTIES
        private bool HuboAlagrue { get; set; }
        public ClasificacionFaseEliminatoria Fase { get; set; }

        //COSNTRUCTOR
        public FaseEliminatoria(DateTime fechaPartido, bool alargue,
ClasificacionFaseEliminatoria fase) : base(fechaPartido)
        {
            HuboAlagrue = alargue;
            Fase = fase;
        }

        //METODOS

        //  --> definicion por penales
        public bool DefinidoPorPenales()
        {
```

```
            bool huboPenales = false;
            foreach (Incidencia item in GetIncidenciasPartido())
            {
                if (item.Minuto.Equals(-1))  //si existio incidencia en min -1 entonces hubo
penales
                {
                    huboPenales = true;
                    break;
                }
            }
            return huboPenales;
        }

        //  --> mensaje de ganador
        public override string MensajeFinalPartido()
        {
            string mensaje = $"Ganador: {DeterminacionDelGanador()}";

            if (DefinidoPorPenales())
            //el metodo determinacion del ganador ya considera que el partido este terminado
            {
                mensaje = $"Empate en tiempo de juego. Ganador {DeterminacionDelGanador()} en
tanda de penales";
            }
            return mensaje;
        }

        public bool HuboALargue()
        {
            return HuboAlagrue;
        }


        //  --> finalizacion partido
        public override void FinalizacionPartido()
        {
            PartidoFinalizado = true;
            ResultadoFinal = MensajeFinalPartido();

            foreach (Incidencia i in GetIncidenciasPartido())
            {
                if(i.Minuto > 90  || i.Minuto.Equals(-1))
                {
                    HuboAlagrue = true;
                    break;
                }
            }
        }
    }
```

4.4 Clase Selección

```
public class Seleccion : IValidacion, IComparable<Seleccion>
    {
        //PROPERTIES
        private static int AutoIdSeleccion { get; set; } = 1;
        public int IdSeleccion { get; set; }
        public Pais Pais { get; set; }
        private List <Jugador> JugadoresSeleccion { get; set; }

        //CONSTRUCTOR
        public Seleccion(Pais pais)
```

```
        {
            IdSeleccion = AutoIdSeleccion;
            AutoIdSeleccion++;
            Pais = pais;
            JugadoresSeleccion = new List<Jugador>();
        }

        //METODOS

        //  --> agregar jugador a la lista
        public void AgregarJugadorAseleccion(Jugador jugador)
        {
            JugadoresSeleccion.Add(jugador);
        }

        //  --> getJugadores de la selccion
        public List<Jugador> GetJugadoresSeleccion()
        {
            JugadoresSeleccion.Sort();
            return JugadoresSeleccion;
        }

        //  --> validacion
        public void EsValido()
        {
            if(Pais == null || JugadoresSeleccion.Count < 11)
            {
                throw new Exception("La saleccion no posee pais asociado o cantidad de
jugadores menor a 11");
            }
        }

        public int CompareTo([AllowNull] Seleccion other)
        {
            if (Pais.NombrePais.CompareTo(other.Pais.NombrePais) > 0)
            {
                return +1;
            }
            else if (Pais.NombrePais.CompareTo(other.Pais.NombrePais) < 0)
            {
                return -1;
            }
            else
            {
                return 0;
            }
        }
    }
```

4.5 Clase Incidencia

```
public class Incidencia : IValidacion, IComparable<Incidencia>
    {
        //PROPERTIES
        private static int AutoIdIncidencia { get; set; } = 1;
        public int IdIncidencia { get; set; }
        public TipoIncidencia Hecho { get; set; }
        public int Minuto { get; set; }
        public int JugadorId { get; set; }

        //CONSTRUCTOR
        public Incidencia(int jugador, TipoIncidencia hecho, int minuto)
```

```
        {
            IdIncidencia = AutoIdIncidencia;
            AutoIdIncidencia++;
            JugadorId = jugador;
            Hecho = hecho;
            Minuto = minuto;
        }

        //METODO

        //  --> validacion de incidencia
        public void EsValido()
        {
            if (JugadorId <= 0 || JugadorId > 871 || Minuto < -1)
            {
                throw new Exception("La incidencia no es valida");
            }
        }

        //  --> compare to
        public int CompareTo([AllowNull] Incidencia other)
        {
            if (Minuto.CompareTo(other.Minuto) > 0)
            {
                return +1;
            }
            else if (Minuto.CompareTo(other.Minuto) < 0)
            {
                return -1;
            }
            else
            {
                return 0;
            }

        }
    }
```

4.6 Clase Jugador

```
public class Jugador: IValidacion, IComparable<Jugador>
    {
        //PROPERTIES
        public static double ValorReferencia { get; set; }
        public int IdJugador { get; set; }
        public string NombreCompJugador { get; set; }
        public Pais PaisJugador { get; set; }
        public string NumeroCamisetaJugador { get; set; }
        public DateTime FechaNacJugador { get; set; }
        public double AlturaJugador { get; set; }
        public string PieHabil { get; set; }
        public string PosicionJugador { get; set; }
        public int ValorMercado { get; set; }
        public string MonedaValorMercado { get; set; }

        //CONSTRUCTOR
        public Jugador(int idJugador, string numeroCamisetaJugador, string nombreCompJugador,
DateTime fechaNacJugador, double alturaJugador, string pieHabil, int valorMercado, string
moneda ,Pais paisJugador, string posicionJugador)
        {
            IdJugador = idJugador;
            ValorMercado = valorMercado;
```

```
            NombreCompJugador = nombreCompJugador;
            PaisJugador = paisJugador;
            NumeroCamisetaJugador = numeroCamisetaJugador;
            FechaNacJugador = fechaNacJugador;
            AlturaJugador = alturaJugador;
            PieHabil = pieHabil;
            PosicionJugador = posicionJugador;
            MonedaValorMercado = moneda;
        }

        //METODOS

        //  --> categoriaJugador
        public string CategoriaJugador()
        {
            string categoria = "Estandar";
            if (ValorMercado >= ValorReferencia)
            {
                categoria = "VIP";
            }
            return categoria;
        }


        //  --> validacionJugador
        public void EsValido()
        {
            if (NombreCompJugador == "" || PaisJugador == null || NumeroCamisetaJugador == ""
|| FechaNacJugador == null
                || AlturaJugador < 0 || PieHabil == "" || PosicionJugador == "" ||
ValorMercado < 0)
            {
                throw new Exception("Los datos ingresados para juagdores no son validos");
            }
        }



        //  --> ordenamiento jugadores
        public int CompareTo([AllowNull] Jugador other)
        {
            //ordeno por valor de mercado descendente
            if (ValorMercado.CompareTo(other.ValorMercado) > 0)
            {
                return -1;
            }
            else if (ValorMercado.CompareTo(other.ValorMercado) < 0)
            {
                return 1;
            } else
            {
                //si valor de mercado igual entonces se ordenan por nombre ascendente
                if (NombreCompJugador.CompareTo(other.NombreCompJugador) > 0)
                {
                    return 1;
                }
                else if (NombreCompJugador.CompareTo(other.NombreCompJugador) < 0)
                {
                    return -1;
                }
                else {
                    //si coincide tambien la primer letra? entonces queda como esta en la
posicion
                    return 0; }
            }
        }
    }
```

4.7 Clase País

```
public class Pais:IValidacion
    {
        //PROPERTIES
        private static int AutoIdPais { get; set; } = 1;
        public int IdPais { get; set; }
        public string NombrePais { get; set; }
        public string CodigoPais { get; set; }

        //CONSTRUCTOR
        public Pais(string nombrePais, string codigoPais)
        {
            IdPais = AutoIdPais;
            AutoIdPais++;
            NombrePais = nombrePais;
            CodigoPais = codigoPais;
        }

        //METODOS

        //  --> validacion
        public void EsValido()
        {
            if(NombrePais == "")
            {
                throw new Exception("No se ingresó nombre de pais pais");
            }
            else
            {
                int cantidadLetras = 0;
                // no puedo usar count ya que contaria caracteres e interesa saber letras
                foreach (char item in CodigoPais.ToLower())
                {
                    if ((int)item >= 97 && (int)item <= 122)
                    {
                        cantidadLetras += 1;
                    }
                }
                if (cantidadLetras != 3)
                {
                    throw new Exception("El codigo del país es incorrecto");
                }
            }
        }
    }
```

4.8 Clase Usuario

```
public abstract class Usuario
    {
        //PROPERTIES
        private static int AutoId { get; set; } = 1;
        public int Id { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public string Email { get; set; }
```

```
        public string Password { get; set; }

        //CONSTRUCTOR
        protected Usuario()
        {
            Id = AutoId;
            AutoId++;
        }

        protected Usuario(string nombre, string apellido, string email, string password)
        {
            Id = AutoId;
            AutoId++;
            Nombre = nombre;
            Apellido = apellido;
            Email = email;
            Password = password;
        }
    }
```

4.9 Clase Periodista

```
public class Periodista : Usuario, IValidacion, IComparable<Periodista>
    {

        //CONTRUCTOR
        public Periodista() : base() { }

        public Periodista(string nombre, string apellido, string email, string password) :
base(nombre, apellido, email, password) { }

        //METODOS

        //  --> validacionPeriodista
        public void EsValido()
        {
            if (Nombre == null || Apellido == null || Password == null || Email == null)
            {
                throw new Exception("Alguno de los datos requeridos se encuentra vacío");
            }
            else if (Password.Length < 8)
            {
                throw new Exception("La contraseña debe poseer una longitud mayor a 8
caracteres");
            }
            else if (Email[0] == '@' || Email[Email.Length - 1] == '@')
            {
                throw new Exception("Error: El mail comienza o termina con '@'");
            }
            else
            {
                int cantidadArroba = 0;
                foreach (char item in Email)
                {
                    if (item == '@')
                    {
                        cantidadArroba += 1;
                    }
                }
                if (cantidadArroba != 1)
                {
```

```
                    throw new Exception("El mail ingresado no es valido: no posee o posee mas
de 1 sola @");
                }
            }
        }

        //  --> equals para periodista (para control de creación)
        public override bool Equals(object obj)
        {
            return obj is Periodista periodista && Email == periodista.Email;
        }

        //  --> comparacion de periodistas ordenados ascendentemente
        public int CompareTo([AllowNull] Periodista other)
        {
            if (Apellido.CompareTo(other.Apellido) > 0)
            {
                return +1;
            }
            else if (Apellido.CompareTo(other.Apellido) < 0)
            {
                return -1;
            }
            else
            {
                if (Nombre.CompareTo(other.Nombre) > 0)
                {
                    return +1;
                }
                else if (Nombre.CompareTo(other.Nombre) < 0)
                {
                    return -1;
                }
                else
                {
                    return 0;
                }
            }
        }
    }
```

4.10 Operador

```
public class Operador : Usuario
    {
        //PROPERTIES
        public DateTime FechaInicio { get; set; }

        //CONTRUCTOR
        public Operador() : base() { }

        public Operador(string nombre, string apellido, string email, string password,
DateTime inicio) : base(nombre, apellido, email, password)
        {
            FechaInicio = inicio;
        }

    }
```

4.11 Clase Reseña

```
public class Reseña : IComparable<Reseña>, IValidacion
    {
        //PROPERTIES
        private static int AutoIdReseña { get; set; } = 1;
        public int IdReseña { get; set; }
        public Periodista Periodista { get; set; }
        public DateTime FechaReseña { get; set; }
        public Partido Partido { get; set; }
        public string TituloReseña { get; set; }
        public string ContenidoReseña { get; set; }

        //CONSTRUCTOR
        public Reseña(Periodista periodista, Partido partido, string tituloReseña, string
contenidoReseña)
        {
            IdReseña = AutoIdReseña;
            AutoIdReseña++;
            FechaReseña = DateTime.Today;
            Periodista = periodista;
            Partido = partido;
            TituloReseña = tituloReseña;
            ContenidoReseña = contenidoReseña;
        }

        public Reseña(Periodista periodista, Partido partido, string tituloReseña, string
contenidoReseña, DateTime fecha)
        {
            IdReseña = AutoIdReseña;
            AutoIdReseña++;
            FechaReseña = fecha;
            Periodista = periodista;
            Partido = partido;
            TituloReseña = tituloReseña;
            ContenidoReseña = contenidoReseña;
        }

        public Reseña()
        {
            IdReseña = AutoIdReseña;
            AutoIdReseña++;
            FechaReseña = DateTime.Today;
        }

        public int CompareTo([AllowNull] Reseña other)
        {
            if (FechaReseña.DayOfYear.CompareTo(other.FechaReseña.DayOfYear) > 0)
            {
                return -1;
            }
            else if (FechaReseña.DayOfYear.CompareTo(other.FechaReseña.DayOfYear) < 0)
            {
                return 1;
            }
            else
            {
                if (FechaReseña.Hour.CompareTo(other.FechaReseña.Hour) > 0)
                {
                    return -1;
                }
                else if (FechaReseña.Hour.CompareTo(other.FechaReseña.Hour) < 0)
                {
                    return +1;
                }
                else
```

```
                {
                    return 0;
                }
            }
        }

        public void EsValido()
        {
            if (TituloReseña == null || ContenidoReseña == null)
            {
                throw new Exception("Existen campos vacios. Por favor completelos para
realizar la resena");
            }
        }

        public string GrupoPartido()
        {
            string mensaje = null;

            if (Partido is FaseGrupo)
            {
                FaseGrupo _ret = Partido as FaseGrupo;
                mensaje = _ret.Fase.ToString();
            }
            else if (Partido is FaseEliminatoria)
            {
                FaseEliminatoria _ret = Partido as FaseEliminatoria;
                mensaje = _ret.Fase.ToString();
            }
            return mensaje;
        }
    }
```

4.12 Clase Sistema

```
public class Sistema
  {
    //PROPERTIES
    private List<Usuario> Usuarios { get; set; }
    private List<Pais> Paises { get; set; }
    private List<Jugador> Jugadores { get; set; }
    private List<Seleccion> Selecciones { get; set; }
    private List<Partido> Partidos { get; set; }
    private List<Incidencia> Incidencias { get; set; }
    private List<Reseña> Reseñas { get; set; }

    //CONSTRUCTOR singleton y precarga
    private static Sistema instancia = null;
    private Sistema()
    {
```

```
    Usuarios = new List<Usuario>();
    Paises = new List<Pais>();
    Jugadores = new List<Jugador>();
    Selecciones = new List<Seleccion>();
    Partidos = new List<Partido>();
    Incidencias = new List<Incidencia>();
    Reseñas = new List<Reseña>();

    Precarga();
}
public static Sistema GetSistema()
{
    if (instancia == null)
    {
        instancia = new Sistema();
    }
    return instancia;
}


//METODOS

//RESENAS
//  --> alta y get reseñas
public List<Reseña> GetReseñas()
{
    List<Reseña> _ret = new List<Reseña>();

    foreach (Reseña r in Reseñas)
    {
        if (r.Partido.PartidoFinalizado)
        {
            _ret.Add(r);
        }
    }
```

```
    _ret.Sort();
    return _ret;
}

public List<Reseña> GetReseñasPeriodista(int? id)
{
    List<Reseña> _ret = new List<Reseña>();

    foreach (Reseña r in GetReseñas())
    {
        if (r.Periodista.Id.Equals(id))
        {
            _ret.Add(r);
        }
    }
    _ret.Sort();
    return _ret;
}

public void AltaResena(Reseña r)
{
    try
    {
        r.EsValido();
        Reseñas.Add(r);
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}

// reseñas para un periodista que comento sobre tarjeta roja

public List<Partido> GetPartidosReseñasPeriodistaTRoja(string email)
```

```
{
    List<Partido> _ret = new List<Partido>();

    foreach (Reseña r in GetReseñas())
    {
        if (r.Periodista.Email.Equals(email))
        {
            foreach (Incidencia i in r.Partido.GetIncidenciasPartido())
            {
                if (i.Hecho.ToString().Equals("TarjetaRoja"))
                {
                    _ret.Add(r.Partido);
                    break;
                }
            }
        }
    }
    _ret.Sort();
    return _ret;
}

//USUARIOS
//  --> login

public Usuario LogIn(string usuario, string contraseña)
{
    if (usuario != null || contraseña != null)
    {
        foreach (Usuario u in Usuarios)
        {
            if (u.Email.Equals(usuario.ToLower()))
            {
                if (u.Password.Equals(contraseña))
                {
                    return u;
```

```
            }
            break;
        }
    }

  }
  return null;
}



//PERIODISTAS

//  --> alta y get periodistas

public List<Periodista> GetPeriodistas()
{
  List<Periodista> _ret = new List<Periodista>();
  //casteo
  foreach (Usuario u in Usuarios)
  {
    if (u is Periodista)
    {
      Periodista _p = u as Periodista;
      _ret.Add(_p);
    }
  }
  _ret.Sort();
  return _ret;
}

public void AltaPeriodista(Periodista periodista)
{
  try
  {
```

```
            if (!Usuarios.Contains(periodista))   //esta bien este contains se plantea equals para periodista
            {
                periodista.EsValido();
                Usuarios.Add(periodista);
            }
            else
            {
                throw new Exception("El periodista ya existe en la base de datos");
            }
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
    }

    public void GeneracionPeriodista(string nombrePeriodista, string apellidoPeriodista, string email, string password)
    {
        try
        {
            Periodista p = new Periodista(nombrePeriodista, apellidoPeriodista, email, password);
            if (!Usuarios.Contains(p))  //esta bien este contains se plantea equals para periodista
            {
                AltaPeriodista(p);
            }
            else
            {
                throw new Exception("Periodista ya existe en la base de datos");
            }
        }
        catch (Exception e)
        {
            throw e;
        }
```

```
}

//  --> get periodista segun Id

public Periodista GetPeriodista(int? id)
{
    Periodista _ret = new Periodista();

    foreach (Periodista p in GetPeriodistas())
    {
        if (p.Id.Equals(id))
        {
            _ret = p;
        }
    }
    return _ret;
}

//PAIS
//  --> alta, getPaises, GetPais segun string, get pais de un jugador, get partidos que participo un jugador
public List<Pais> GetPaises()
{
    return Paises;
}

public void AltaPais(Pais pais)
{
    try
    {
        pais.EsValido();
        Paises.Add(pais);
    }
    catch (Exception e)
    {
```

```
        throw new Exception(e.Message);
    }
}

public Pais GetPais(string pais)
{
    Pais paisAretornar = null;

    foreach (Pais item in GetPaises())
    {
        if (item.NombrePais == pais)
        {
            paisAretornar = item;
            break;
        }
    }
    return paisAretornar;
}

public Pais GetPaisDeJugador(int idJugador)
{
    Pais paisDelJugador = null;
    foreach (Jugador item in GetJugadores())
    {
        if (item.IdJugador == idJugador)
        {
            paisDelJugador = item.PaisJugador;
            break;
        }
    }
    return paisDelJugador;
}

public List<Partido> GetPartidosDejugador(int idJugador)
{
```

```
        //no considera que el jugador haya sido expulsado anteriormente
        List<Partido> _ret = new List<Partido>();
        Pais buscado = GetPaisDeJugador(idJugador);
        foreach (Partido partido in GetPartidos())
        {
            foreach (Seleccion seleccion in partido.GetSeleccionesPartido())
            {
                if (seleccion.Pais.NombrePais == buscado.NombrePais)
                {
                    _ret.Add(partido);
                    break;
                }
            }
        }
        return _ret;
    }



    //JUGADOR
    //  --> alta, getJugadores, get jugadores de una seleccion, get jugador segun ID, get jugador expulsado ordenado, get jugador que convirtio al menos un gol
    public List<Jugador> GetJugadores()
    {
        Jugadores.Sort();
        return Jugadores;
    }


    public void AltaJugador(Jugador jugador)
    {
        try
        {
            jugador.EsValido();
            Jugadores.Add(jugador);
        }
        catch (Exception e)
```

```
    {
        throw new Exception(e.Message);
    }
}

private List<Jugador> JugadoresDe(Pais pais)
{
    List<Jugador> _misJugadores = new List<Jugador>();
    foreach (Jugador item in Jugadores)
    {
        if (item.PaisJugador.NombrePais.Equals(pais.NombrePais))
        {
            _misJugadores.Add(item);
        }
    }
    _misJugadores.Sort();
    return _misJugadores;
}

public Jugador GetJugadorSegunId(int idJugador)
{
    Jugador buscado = null;
    foreach (Jugador item in GetJugadores())
    {
        if (item.IdJugador == idJugador)
        {
            buscado = item;
            break;
        }
    }
    return buscado;
}

public List<Jugador> GetJugadoresExpulsados()
{
```

```
    List<Jugador> _ret = new List<Jugador>();
    foreach (Incidencia item in GetIncidencias())
    { //evaluo que el jugador expulsado no se repita
        if (item.Hecho == TipoIncidencia.TarjetaRoja && !_ret.Contains(GetJugadorSegunId(item.JugadorId)))
        {
            _ret.Add(GetJugadorSegunId(item.JugadorId));
        }
    }
    _ret.Sort();
    return _ret;
}

public List<Jugador> GetJugadoresRealizaronGol()
{
    List<Jugador> _ret = new List<Jugador>();
    foreach (Incidencia item in GetIncidencias())
    {
        if (item.Hecho == TipoIncidencia.Gol && !_ret.Contains(GetJugadorSegunId(item.JugadorId)))
        {
            _ret.Add(GetJugadorSegunId(item.JugadorId));
        }
    }
    _ret.Sort();
    return _ret;
}

//asignar valor de referencia
public void AsignarValorReferencia(double valor)
{
    Jugador.ValorReferencia = valor;
}



//get jugadores de un partido
```

```
public List<Jugador> GetJugadoresDePartidoConIncidencias(int id)
{
    Partido p = GetPartido(id);

    List<Jugador> jugadoresPartido = new List<Jugador>();

    foreach (Incidencia i in p.GetIncidenciasPartido())
    {
        if (!jugadoresPartido.Contains(GetJugadorSegunId(i.JugadorId)))
        {
            jugadoresPartido.Add(GetJugadorSegunId(i.JugadorId));
        }
    }
    return jugadoresPartido;
}

//SELECCION
//  --> alta, getSelecciones, get seleccion para un partido, get seleccion con la que jugo otra seleccion
public List<Seleccion> GetSelecciones()
{
    Selecciones.Sort();
    return Selecciones;
}

public void AltaSeleccion()
{
    try
    {
        foreach (Pais p in Paises)
        {
            // Se crea una seleccion por cada país en la lista.
            Seleccion seleccion = new Seleccion(p);
            List<Jugador> jugadoresDeSel = JugadoresDe(p);
            foreach (Jugador jugador in jugadoresDeSel)
            {
```

```
                seleccion.AgregarJugadorAseleccion(jugador);
            }
            seleccion.EsValido();
            Selecciones.Add(seleccion);
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}

public Seleccion GetSeleccionParaPartido(string pais)
//devuelve el objeto seleccion a partir de un string
{
    Seleccion ret = null;
    foreach (Seleccion item in GetSelecciones())
    {
        if (item.Pais.NombrePais == pais)
        {
            ret = item;
            break;
        }
    }
    return ret;
}

public Seleccion GetSeleccionQueJugoContra(Partido partido, Pais pais)
{
    Seleccion buscada = null;
    foreach (Seleccion item in partido.GetSeleccionesPartido())
    {
        if (item.Pais.NombrePais != pais.NombrePais)
        {
            buscada = item;
```

```
            break;
        }
    }
    return buscada;
}


//get seleccion con mas goles

public List<Seleccion> GetSeleccionesConMasGoles()
{
    int countMaximo = 0;
    List<Seleccion> s = new List<Seleccion>();

    foreach (Seleccion item in GetSelecciones())
    {
        int count = 0;
        foreach (Partido p in GetPartidos())
        {
            if (p.PartidoFinalizado)
            {
                foreach (Incidencia i in p.GetIncidenciasPartido())
                {
                    if (GetJugadorSegunId(i.JugadorId).PaisJugador.Equals(item.Pais)
                        && i.Hecho.ToString().Equals("Gol") && i.Minuto != -1)
                    {
                        count++;
                    }
                }
            }
        }
        if (count > countMaximo)
        {
            countMaximo = count;
            s.Clear();
```

```
            s.Add(item);
        }
        else if (count == countMaximo && countMaximo != 0)
        {
            s.Add(item);
        }
    }
    return s;
}


//PARTIDOS
//  --> alta, get partidos, get partidos que participo una seleccion y partidos con mayor cantidad de goles
public List<Partido> GetPartidos()
{
    return Partidos;
}

public Partido GetPartido(int id)
{
    Partido _ret = null;
    foreach (Partido p in GetPartidos())
    {
        if (p.IdPartido.Equals(id))
        {
            _ret = p;
            break;
        }
    }
    return _ret;
}

public List<Partido> GetPartidosFinalizados()
{
    List<Partido> _ret = new List<Partido>();
```

```
    foreach (Partido p in GetPartidos())
    {
        if (p.PartidoFinalizado)
        {
            _ret.Add(p);
        }
    }
    _ret.Sort();
    return _ret;
}

public List<Partido> GetPartidosFinalizadosEntre(DateTime d1, DateTime d2)
{
    List<Partido> _ret = new List<Partido>();
    foreach (Partido p in GetPartidosFinalizados())
    {
        if (p.FechaPartido >= d1 && p.FechaPartido <= d2)
        {
            _ret.Add(p);
        }
    }
    _ret.Sort();
    return _ret;

}



public List<Partido> GetPartidosNoFinalizados()
{
    List<Partido> _ret = new List<Partido>();
    foreach (Partido p in GetPartidos())
    {
        if (!p.PartidoFinalizado)
        {
            _ret.Add(p);
```

```
        }
    }
    _ret.Sort();
    return _ret;
}

public void AltaPartido(Partido partido)
{
    try
    {
        partido.EsValido();
        Partidos.Add(partido);
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}

public List<Partido> GetPartidosDeSeleccion(string seleccion)
{
    //se obtienen los partidos jugados por una seleccion
    List<Partido> _ret = new List<Partido>();

    foreach (Partido item in GetPartidos())
    {
        foreach (Seleccion item2 in item.GetSeleccionesPartido())
        {
            if (item2.Equals(GetSeleccionParaPartido(seleccion)))
            {
                _ret.Add(item);
                break;
            }
        }
    }
```

```
        return _ret;
    }

    public List<Partido> GetPartidoConMasGoles(string seleccion)
    {
        //busco para una seleccion cual fue el partido con mas goles
        List<Partido> _ret = new List<Partido>();

        int golesMaximos = 0;  //esta variable va registrando cuantos goles maximos se van realizado por partido

        foreach (Partido item in GetPartidosDeSeleccion(seleccion))
        {
            int goles = 0;  //variable que analiza cuantos goles tiene el partido
            foreach (Incidencia item2 in item.GetIncidenciasPartido())
            {
                if (GetJugadorSegunId(item2.JugadorId).PaisJugador.NombrePais == seleccion && item2.Hecho ==
TipoIncidencia.Gol
                    && item2.Minuto != -1)
                {
                    goles += 1;
                }
            }
            if (goles == golesMaximos && golesMaximos != 0)
            {
                _ret.Add(item);
            }
            else if (goles > golesMaximos)
            {
                golesMaximos = goles;
                _ret.Clear();
                //si los goles realizados superan a los registrados entonces se vacia la lista (es lista para considerar si
existen partidos con
                //igual cantidad de goles
                _ret.Add(item);
            }
        }
```

```
        return _ret;
    }



    //INCIDENCIAS
    //  --> get y alta incidencia
    public List<Incidencia> GetIncidencias()
    {
        return Incidencias;
    }

    public void AltaIncidencia(Incidencia incidencia)
    {
        try
        {
            incidencia.EsValido();
            Incidencias.Add(incidencia);
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
    }
```

```
//CREACION DE PERIODISTA
                Periodista Periodista1 = new Periodista("Joaquin", "Cardozo", "jc@gmail.com",
"123456789");
                AltaPeriodista(Periodista1);

                Periodista Periodista2 = new Periodista("Javier", "Fernandez",
"jf@gmail.com", "123456789");
                AltaPeriodista(Periodista2);

                Periodista Periodista3 = new Periodista("Andres", "Fernandez",
"af@gmail.com", "123456789");
                AltaPeriodista(Periodista3);

                //CREACION DE OPERADOR
                Operador Operador1 = new Operador("Emanuel", "Gonzalez", "eg@gmail.com",
"123456789", new DateTime(2012, 11, 10));
                Usuarios.Add(Operador1);
```

```
                Operador Operador2 = new Operador("Gonzalo", "Fernandez", "gf@gmail.com",
"123456789", new DateTime(2011, 11, 10));
                Usuarios.Add(Operador2);


//CREACION DE PARTIDOS


                //   ---->  Fase A


                FaseGrupo partido1A = new FaseGrupo(new DateTime(2022, 11, 21, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido1A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Argentina"),
GetSeleccionParaPartido("Catar"));

                AltaPartido(partido1A);


                FaseGrupo partido2A = new FaseGrupo(new DateTime(2022, 11, 22, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido2A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Argentina"),
GetSeleccionParaPartido("Alemania"));

                AltaPartido(partido2A);


                FaseGrupo partido3A = new FaseGrupo(new DateTime(2022, 11, 23, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido3A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Alemania"),
GetSeleccionParaPartido("Dinamarca"));

                AltaPartido(partido3A);


                FaseGrupo partido4A = new FaseGrupo(new DateTime(2022, 11, 24, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido4A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Catar"),
GetSeleccionParaPartido("Alemania"));

                AltaPartido(partido4A);


                FaseGrupo partido5A = new FaseGrupo(new DateTime(2022, 11, 25, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido5A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Catar"),
GetSeleccionParaPartido("Dinamarca"));

                AltaPartido(partido5A);


                FaseGrupo partido6A = new FaseGrupo(new DateTime(2022, 11, 26, 18, 00, 00),
ClasificacionFaseGrupo.A);

                partido6A.AgregarSeleccionesPartido(GetSeleccionParaPartido("Argentina"),
GetSeleccionParaPartido("Dinamarca"));

                AltaPartido(partido6A);
```

```
            //   ---->  Fase B


            FaseGrupo partido1B = new FaseGrupo(new DateTime(2022, 11, 21, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido1B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Brasil"),
GetSeleccionParaPartido("Croacia"));

            AltaPartido(partido1B);


            FaseGrupo partido2B = new FaseGrupo(new DateTime(2022, 11, 22, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido2B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Francia"),
GetSeleccionParaPartido("Bélgica"));

            AltaPartido(partido2B);


            FaseGrupo partido3B = new FaseGrupo(new DateTime(2022, 11, 23, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido3B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Bélgica"),
GetSeleccionParaPartido("Croacia"));

            AltaPartido(partido3B);


            FaseGrupo partido4B = new FaseGrupo(new DateTime(2022, 11, 24, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido4B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Francia"),
GetSeleccionParaPartido("Croacia"));

            AltaPartido(partido4B);


            FaseGrupo partido5B = new FaseGrupo(new DateTime(2022, 11, 25, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido5B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Brasil"),
GetSeleccionParaPartido("Francia"));

            AltaPartido(partido5B);


            FaseGrupo partido6B = new FaseGrupo(new DateTime(2022, 11, 26, 14, 00, 00),
ClasificacionFaseGrupo.B);

            partido6B.AgregarSeleccionesPartido(GetSeleccionParaPartido("Brasil"),
GetSeleccionParaPartido("Bélgica"));

            AltaPartido(partido6B);


            //   ---->  Fase Octavos de Final


            FaseEliminatoria partido1SF = new FaseEliminatoria(new DateTime(2022, 11, 27, 13,
30, 00), true, ClasificacionFaseEliminatoria.Octavos);
```

```
                partido1SF.AgregarSeleccionesPartido(GetSeleccionParaPartido("Brasil"),
GetSeleccionParaPartido("Argentina"));

                AltaPartido(partido1SF);


                FaseEliminatoria partido2SF = new FaseEliminatoria(new DateTime(2022, 11, 27, 18,
30, 00), true, ClasificacionFaseEliminatoria.Octavos);

                partido2SF.AgregarSeleccionesPartido(GetSeleccionParaPartido("Francia"),
GetSeleccionParaPartido("Alemania"));

                AltaPartido(partido2SF);



                //INCIDENCIAS


                //   ---->  Fase A


                Incidencia I1 = new Incidencia(31, TipoIncidencia.Gol, 30);

                AltaIncidencia(I1);

                partido1A.ValidarYagregarIncidencia(I1);

                Incidencia I2 = new Incidencia(52, TipoIncidencia.TarjetaAmarilla, 59);

                AltaIncidencia(I2);

                partido1A.ValidarYagregarIncidencia(I2);

                Incidencia I3 = new Incidencia(9, TipoIncidencia.Gol, 21);

                AltaIncidencia(I3);

                partido1A.ValidarYagregarIncidencia(I3);

                Incidencia I4 = new Incidencia(9, TipoIncidencia.Gol, 55);

                AltaIncidencia(I4);

                partido1A.ValidarYagregarIncidencia(I4);

                Incidencia I5 = new Incidencia(3, TipoIncidencia.TarjetaAmarilla, 60);

                AltaIncidencia(I5);

                partido1A.ValidarYagregarIncidencia(I5);


                Incidencia I6 = new Incidencia(10, TipoIncidencia.Gol, 12);

                AltaIncidencia(I6);

                partido2A.ValidarYagregarIncidencia(I6);

                Incidencia I7 = new Incidencia(58, TipoIncidencia.TarjetaRoja, 49);

                AltaIncidencia(I7);

                partido2A.ValidarYagregarIncidencia(I7);


                Incidencia I8 = new Incidencia(79, TipoIncidencia.Gol, 36);
```

```
AltaIncidencia(I8);
partido3A.ValidarYagregarIncidencia(I8);
Incidencia I9 = new Incidencia(72, TipoIncidencia.Gol, 60);
AltaIncidencia(I9);
partido3A.ValidarYagregarIncidencia(I9);
Incidencia I10 = new Incidencia(84, TipoIncidencia.Gol, 72);
AltaIncidencia(I10);
partido3A.ValidarYagregarIncidencia(I10);

Incidencia I11 = new Incidencia(54, TipoIncidencia.TarjetaAmarilla, 56);
AltaIncidencia(I11);
partido4A.ValidarYagregarIncidencia(I11);
Incidencia I12 = new Incidencia(72, TipoIncidencia.Gol, 80);
AltaIncidencia(I12);
partido4A.ValidarYagregarIncidencia(I12);

Incidencia I13 = new Incidencia(32, TipoIncidencia.Gol, 55);
AltaIncidencia(I13);
partido5A.ValidarYagregarIncidencia(I13);
Incidencia I14 = new Incidencia(31, TipoIncidencia.TarjetaRoja, 73);
AltaIncidencia(I14);
partido5A.ValidarYagregarIncidencia(I14);
Incidencia I15 = new Incidencia(100, TipoIncidencia.TarjetaRoja, 80);
AltaIncidencia(I15);
partido5A.ValidarYagregarIncidencia(I15);

Incidencia I16 = new Incidencia(5, TipoIncidencia.Gol, 32);
AltaIncidencia(I16);
partido6A.ValidarYagregarIncidencia(I16);
Incidencia I17 = new Incidencia(9, TipoIncidencia.Gol, 63);
AltaIncidencia(I17);
partido6A.ValidarYagregarIncidencia(I17);
Incidencia I18 = new Incidencia(5, TipoIncidencia.TarjetaAmarilla, 89);
AltaIncidencia(I18);
partido6A.ValidarYagregarIncidencia(I18);
Incidencia I19 = new Incidencia(92, TipoIncidencia.TarjetaAmarilla, 12);
AltaIncidencia(I19);
partido6A.ValidarYagregarIncidencia(I19);
```

```
//   ---->  Fase B

Incidencia I20 = new Incidencia(110, TipoIncidencia.Gol, 22);
AltaIncidencia(I20);
partido1B.ValidarYagregarIncidencia(I20);
Incidencia I21 = new Incidencia(106, TipoIncidencia.Gol, 62);
AltaIncidencia(I21);
partido1B.ValidarYagregarIncidencia(I21);
Incidencia I22 = new Incidencia(123, TipoIncidencia.Gol, 89);
AltaIncidencia(I22);
partido1B.ValidarYagregarIncidencia(I22);
Incidencia I23 = new Incidencia(181, TipoIncidencia.TarjetaAmarilla, 22);
AltaIncidencia(I23);
partido1B.ValidarYagregarIncidencia(I23);
Incidencia I24 = new Incidencia(192, TipoIncidencia.TarjetaAmarilla, 55);
AltaIncidencia(I24);
partido1B.ValidarYagregarIncidencia(I24);

Incidencia I25 = new Incidencia(144, TipoIncidencia.TarjetaAmarilla, 65);
AltaIncidencia(I25);
partido2B.ValidarYagregarIncidencia(I25);

Incidencia I26 = new Incidencia(167, TipoIncidencia.TarjetaRoja, 34);
AltaIncidencia(I26);
partido3B.ValidarYagregarIncidencia(I26);

Incidencia I27 = new Incidencia(150, TipoIncidencia.Gol, 14);
AltaIncidencia(I27);
partido4B.ValidarYagregarIncidencia(I27);
Incidencia I28 = new Incidencia(150, TipoIncidencia.Gol, 65);
AltaIncidencia(I28);
partido4B.ValidarYagregarIncidencia(I28);
Incidencia I29 = new Incidencia(142, TipoIncidencia.Gol, 22);
AltaIncidencia(I29);
partido4B.ValidarYagregarIncidencia(I29);
Incidencia I30 = new Incidencia(186, TipoIncidencia.TarjetaAmarilla, 43);
AltaIncidencia(I30);
```

```
partido4B.ValidarYagregarIncidencia(I30);

Incidencia I31 = new Incidencia(110, TipoIncidencia.Gol, 22);
AltaIncidencia(I31);
partido5B.ValidarYagregarIncidencia(I31);

Incidencia I32 = new Incidencia(130, TipoIncidencia.Gol, 55);
AltaIncidencia(I32);
partido6B.ValidarYagregarIncidencia(I32);
Incidencia I34 = new Incidencia(158, TipoIncidencia.TarjetaAmarilla, 29);
AltaIncidencia(I34);
partido6B.ValidarYagregarIncidencia(I34);

//   ---->  Fase Octavos de final

Incidencia I35 = new Incidencia(10, TipoIncidencia.Gol, 12);
AltaIncidencia(I35);
partido1SF.ValidarYagregarIncidencia(I35);
Incidencia I36 = new Incidencia(18, TipoIncidencia.Gol, -1);
AltaIncidencia(I36);
partido1SF.ValidarYagregarIncidencia(I36);
Incidencia I37 = new Incidencia(25, TipoIncidencia.Gol, -1);
AltaIncidencia(I37);
partido1SF.ValidarYagregarIncidencia(I37);
Incidencia I38 = new Incidencia(3, TipoIncidencia.Gol, -1);
AltaIncidencia(I38);
partido1SF.ValidarYagregarIncidencia(I38);
Incidencia I39 = new Incidencia(1, TipoIncidencia.TarjetaAmarilla, 22);
AltaIncidencia(I39);
partido1SF.ValidarYagregarIncidencia(I39);
Incidencia I40 = new Incidencia(110, TipoIncidencia.Gol, 43);
AltaIncidencia(I40);
partido1SF.ValidarYagregarIncidencia(I40);
Incidencia I41 = new Incidencia(105, TipoIncidencia.Gol, -1);
AltaIncidencia(I41);
partido1SF.ValidarYagregarIncidencia(I41);
Incidencia I42 = new Incidencia(106, TipoIncidencia.Gol, -1);
AltaIncidencia(I42);
```

```
            partido1SF.ValidarYagregarIncidencia(I42);
            Incidencia I43 = new Incidencia(107, TipoIncidencia.Gol, -1);
            AltaIncidencia(I43);
            partido1SF.ValidarYagregarIncidencia(I43);
            Incidencia I44 = new Incidencia(108, TipoIncidencia.Gol, -1);
            AltaIncidencia(I44);
            partido1SF.ValidarYagregarIncidencia(I44);
            Incidencia I45 = new Incidencia(130, TipoIncidencia.TarjetaRoja, 25);
            AltaIncidencia(I45);
            partido1SF.ValidarYagregarIncidencia(I45);

            Incidencia I46 = new Incidencia(150, TipoIncidencia.Gol, 19);
            AltaIncidencia(I46);
            partido2SF.ValidarYagregarIncidencia(I46);
            Incidencia I47 = new Incidencia(79, TipoIncidencia.Gol, 41);
            AltaIncidencia(I47);
            partido2SF.ValidarYagregarIncidencia(I47);
            Incidencia I48 = new Incidencia(72, TipoIncidencia.Gol, 65);
            AltaIncidencia(I48);
            partido2SF.ValidarYagregarIncidencia(I48);
            Incidencia I49 = new Incidencia(132, TipoIncidencia.TarjetaAmarilla, 25);
            AltaIncidencia(I49);
            partido2SF.ValidarYagregarIncidencia(I49);
            Incidencia I50 = new Incidencia(56, TipoIncidencia.TarjetaAmarilla, 10);
            AltaIncidencia(I50);
            partido2SF.ValidarYagregarIncidencia(I50);


            //FINALIZACION PARTIDO

            foreach (Partido item in GetPartidos())
            {
                if (item.IdPartido % 2 == 0 || item.IdPartido % 3 == 0 || item.IdPartido % 5 ==
0)
                {
                    item.FinalizacionPartido();
                }
            }
```

```
                Reseña R1 = new Reseña(Periodista1, partido1A, "LA QUE PRIMERO SALIÓ
CAMPEONA", "Si bien nosotros conocemos el potencial que ensta. Sino más ", new DateTime(2022,
12, 01));
                AltaResena(R1);

                Reseña R2 = new Reseña(Periodista1, partido2SF, "Union Berlín resiste en el
liderato de la Bundesliga", "Aferrado a la épica, con u ventaja cuando el suizo Nico Elvedi
batió a Frederik Ronnow tras recibir un balón de Lars Stindl.Fischer movió el banquillo. Hizo
tres cambios de una tacada. Sacó a Sven Michel, a Genki Haraguchi y a Christopher Trimmel a
la hora de juego. Pero no logró empatar hasta el 79, cuando Nico Elvedi llevó a la red un
centro de Lars Stindl.", new DateTime(2022, 12, 02));
                AltaResena(R2);

                Reseña R3 = new Reseña(Periodista1, partido1SF, "Union Berlín resiste en el
liderato de la Bundesliga", "Aferrado a la épica, con una remontada cu", new DateTime(2022,
12, 03));
                AltaResena(R3);

                Reseña R4 = new Reseña(Periodista2, partido2SF, "Union Berlín resiste en el
liderato de la Bundesliga", "Aferrado a la épica, con una remontada culmia los locales a la
media a", new DateTime(2022, 12, 03));

                Reseña R5 = new Reseña(Periodista3, partido2SF, "Union Berlín resiste en el
liderato de la Bundesliga", "Aferrado a la épica, con una remontada culminada en el minuto
97, ", new DateTime(2022, 12, 03));
                AltaResena(R5);
```

4.13 Enumerado ClasificacionFaseEliminatoria

```
public enum ClasificacionFaseEliminatoria
    {
        Octavos = 0,
        Cuartos = 1,
        Semifinal = 2,
        Final = 3,
    }
```

4.14 Enumerado ClasificacionFaseGrupo

```
public enum ClasificacionFaseGrupo
    {
        A = 0,
        B = 1,
        C = 2,
        D = 3,
        E = 4,
        F = 5,
        G = 6,
        H = 7,
    }
```

4.15 Enumerado TipoIncidencia

```
public enum TipoIncidencia
    {
        TarjetaAmarilla = 0,
        TarjetaRoja = 1,
        Gol = 2,
    }
```

4.16 Interface IValidacion

```
public interface IValidacion
    {
        public void EsValido();
    }
```