

PROYECTO

APUESTAS ONLINE



GRUPO DE:

VERÓNICA HIDALGO

FECHA DE INICIO:

17/12/2021

ACADEMIA

INTERNATIONAL SCHOOL

ÍNDICE

1. DOCUMENTOS LEGALES Pág 4

2. INTRODUCCIÓN Pág 6

2.1 Integrantes del Equipo

2.2 Planificación del Proyecto

2.3 Esquema de las tareas



3. LAS TRES LEYES DE LA COPIA DE SEGURIDAD Pág 8

3.1 Ley 1: Backup Periódico

3.2 Ley 2: Tener una copia de seguridad fuera y en el sitio

3.3 Ley 3: Automatización

4. COPIAS DE SEGURIDAD DESDE EL PROPIO PROGRAMA ... Pág 9

4.1 Respaldo de Base de Datos

4.2 Restauración de Base de Datos

4.3 Respaldo y restauración de MYSQL en Java

4.3.1 Respaldo base de datos de MYSQL en Java

4.3.2 Restaurar base de datos de MYSQL en Java

4.3.3 Proceso para mostrar en pantalla los errores que envía MYSQL

5. ENCAPSULACIÓN DE NUESTRA APLICACIÓN Pág 13

5.1 Definición de Encapsulación Java

5.1.1 Datos

6. INSTALACIÓN Pág 14

7. PROCEDIMIENTOS, FUNCIONES Y DISPARADORES Pág 14

7.1 Procedimientos Almacenados

7.2 Funciones Almacenadas

7.3 Disparadores

8. ESTRUCTURA Pág 15

8.1 Creación de Tablas

8.1.1 La tabla de Usuarios

8.1.2 La tabla de Equipos

8.1.3 La tabla de Apuestas

8.1.4 La tabla de Previsión

8.1.5 Asignación de Variables



9. DISEÑO DE LA ESTRUCTURA Pág 18

10. DISEÑO DE LA APLICACIÓN Pág 19

10.1 Diseño de Logo

10.2 Diseño General de la Aplicación

10.3 Elementos de la Aplicación

10.4 Elementos que Incluiremos en el Futuro

11. REDACCIÓN DE LA INFORMACIÓN Pág 25

11.1 Información Sobre la Liga

11.2 Información Sobre los Equipos

12. COMPONENTES DE LA APLICACIÓN Pág 30

12.1 Lista de Componentes (Imports)

12.2 Import JAVA.SQL*

12.3 Import JAVA.TIME

12.4 Import JAVA.UTIL.LOGGING

12.5 Import JAVA.AWT.IMAGE

12.6 Import JAVA.RMI.REGISTRY

12.7 Import JAVA.AWT.EVENT

12.8 Import JAVA.SWING

12.9 Import JAVA.IMAGE.IO

12.10 Import JAVA.IO

12.11 Importa JAVA.AWT.PRINT

13. LENGUAJE HTML Y CONEXIONES CON EL SERVIDOR Pág 39

14. JAVA SERVERS PAGES (JSP) Pág 41

15. CONEXIÓN BASE DE DATOS Pág 42

16. CLASES MODELO EN JAVA Pág 44

17. CLASES SERVLETS EN JAVA Pág 45

18. NORMAS DE PROGRAMACIÓN Pág 46

19. DISEÑO FINAL DE LA WEB Pág 74

1. DOCUMENTOS LEGALES

Copyright © 2021, 2025. Apuestas Deportivas.

Este software y la documentación relacionada se proporcionan bajo un contrato de licencia que contiene restricciones de uso y divulgación y están protegidos por las leyes de propiedad intelectual. Salvo que esté expresamente permitido en su contrato de licencia o lo permita la ley, no puede usar, copiar, reproducir, traducir, difundir, modificar, licenciar, transmitir, distribuir, exhibir, ejecutar, publicar o mostrar en cualquier parte, en cualquier forma, o por cualquier medio. Se prohíbe la ingeniería inversa, el desmontaje o la descompilación de este software, a menos que lo exija la ley para la interoperabilidad.

La información contenida en este documento está sujeta a cambios sin previo aviso y no se garantiza que esté libre de errores. Si encuentra algún error, infórmenos por escrito.

Si se trata de software o documentación relacionada que se entrega al gobierno de ESPAÑA Oa cualquier persona que lo otorgue en nombre del gobierno de los ESPAÑA, Entonces se aplicará el siguiente aviso:

USUARIOS FINALES DEL GOBIERNO DE ESPAÑA: Programas de Oracle (incluido cualquier sistema operativo, software integrado, cualquier programa incrustado, instalado o activado en el hardware entregado, y modificaciones de dichos programas) y documentación informática de Oracle u otros datos de Oracle entregados a los usuarios finales del gobierno de los ESPAÑA son "software informático comercial" o "documentación de software informático comercial" de conformidad con el Reglamento Federal de Adquisiciones aplicable y los reglamentos suplementarios específicos de la agencia. Como tal, el uso, reproducción,

duplicación, publicación, exhibición, divulgación, modificación, preparación de trabajos derivados y / o adaptación de i) programas de Oracle (incluido cualquier sistema operativo, software integrado, cualquier programa incrustado, instalado o activado en la entrega hardware y modificaciones de dichos programas), ii) la documentación informática de Oracle y / o iii) otros datos de Oracle, están sujetos a los derechos y limitaciones especificados en la licencia contenida en el contrato correspondiente. Los términos que rigen el uso de los servicios en la nube de Oracle por parte del gobierno de ESPAÑA. Se definen en el contrato aplicable para dichos servicios. No se otorgan otros derechos al gobierno de los ESPAÑA.

Este software o hardware está desarrollado para uso general en una variedad de aplicaciones de administración de información. No está desarrollado ni diseñado para su uso en ninguna aplicación intrínsecamente peligrosa, incluidas las aplicaciones que pueden crear un riesgo de lesiones personales. Si usa este software o hardware en aplicaciones peligrosas, entonces será responsable de tomar todas las medidas apropiadas a prueba de fallas, respaldo, redundancia y otras para garantizar su uso seguro. Oracle Corporation y sus afiliadas renuncian a cualquier responsabilidad por los daños causados por el uso de este software o hardware en aplicaciones peligrosas.

Oracle y Java son marcas comerciales registradas de Oracle y / o sus afiliadas. Otros nombres pueden ser marcas comerciales de sus respectivos propietarios.

Intel e Intel Inside son marcas comerciales o marcas comerciales registradas de Intel Corporation. Todas las marcas comerciales de SPARC se utilizan bajo licencia y son marcas comerciales o marcas comerciales registradas de SPARC International, Inc. AMD, Epyc y el logotipo de AMD son marcas comerciales o marcas comerciales registradas de Advanced Micro Devices. UNIX es una marca registrada de The Open Group.

Este software o hardware y documentación pueden proporcionar acceso o información sobre contenido, productos y servicios de terceros. Oracle Corporation y sus afiliadas no son responsables y renuncian expresamente a todas las garantías de ningún tipo con respecto al contenido, productos y servicios de terceros, a menos que se establezca lo contrario en un acuerdo aplicable entre usted y Oracle. Oracle Corporation y sus afiliadas no serán responsables de ninguna pérdida, costo o daño incurrido debido a su acceso o uso de contenido, productos o servicios de terceros, excepto según lo establecido en un acuerdo aplicable entre usted y Oracle.

Esta documentación NO se distribuye bajo una licencia GPL. El uso de esta documentación está sujeto a los siguientes términos:

Puede crear una copia impresa de esta documentación únicamente para su uso personal. Se permite la conversión a otros formatos siempre que el contenido real no se altere o edite de ninguna manera. No debe publicar ni distribuir esta documentación de ninguna forma ni en ningún medio, excepto si distribuye la documentación de una manera similar a como la distribuye Oracle (es decir, electrónicamente para descargar en un sitio web con el software) o en un CD, -ROM o medio similar, siempre que la documentación se difunda junto con el software en el mismo medio. Cualquier otro uso, como la difusión de copias impresas o el uso de esta documentación, total o parcialmente, en otra publicación, requiere el consentimiento previo por escrito de un representante autorizado de Oracle. Oracle y / o sus afiliadas se reservan todos y cada uno de los derechos sobre esta documentación que no se hayan otorgado expresamente anteriormente.

2. INTRODUCCIÓN

La **realización del proyecto en grupo** consiste en la creación de un programa enfocado en las apuestas deportivas online, más concretamente apuestas futbolísticas, programado en Java mediante la aplicación de escritorio NetBeans (Oracle Corporation).

El programa realiza pronósticos sobre el resultado de los **partidos emitidos en directo**, incluido en el programa previamente establecido por nuestro equipo.

2.1 Integrantes del Equipo

Los **integrantes que conforman el equipo** son: Patricia López Sánchez, Mark Abarca Adrio, Verónica Hidalgo Ramírez, Noah Ezquerra Contioso, Javier Palacios Botejara y Sandra López Gallego.

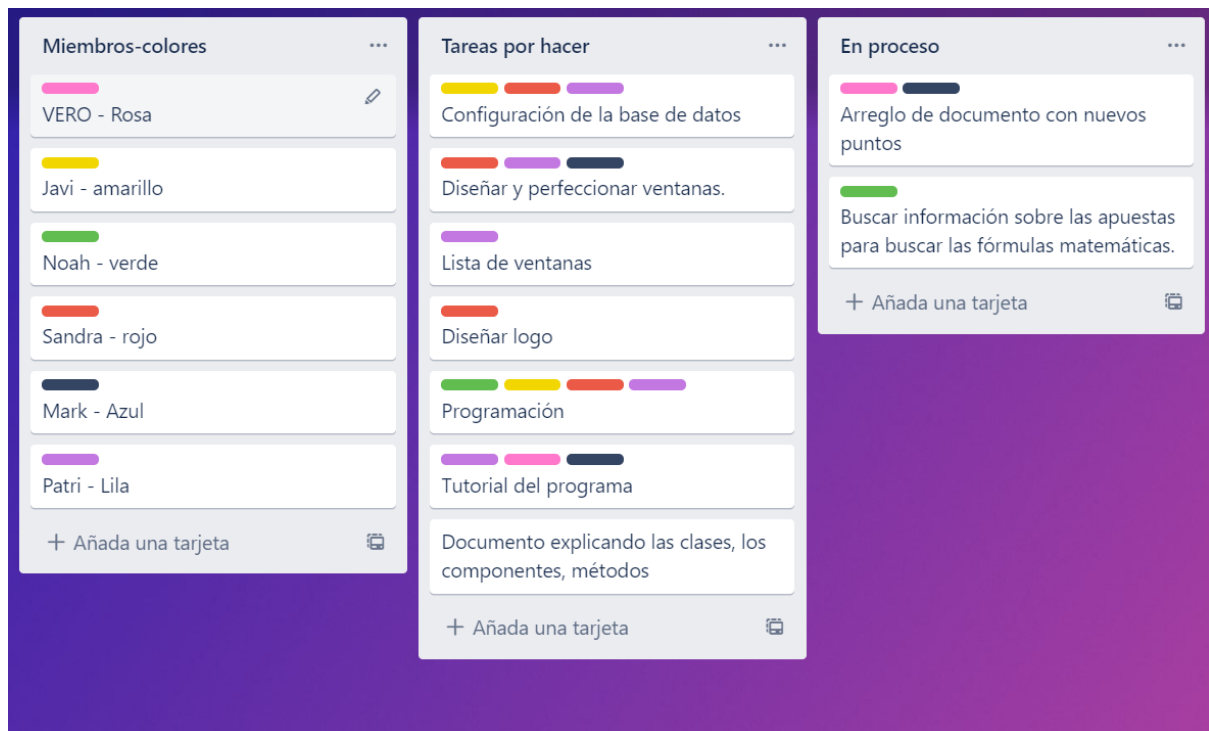
2.2 Planificación de Proyecto

La **estructuración de las tareas** a realizar en la duración del proyecto es la siguiente:

- Elaboración de diseño de la aplicación (Sandra López)
- Redacción del proyecto y búsqueda de fuentes (Patricia López, Mark Abarca, Verónica Hidalgo)
- Programación de la aplicación (Noah Ezquerra, Javier Palacios)

2.3 Esquema de las tareas.

En este esquema se ordena las tareas diarias realizadas, ideas y a quien se les asignan.



3. LAS 3 LEYES DE LA COPIA DE SEGURIDAD

Para realizar una copia de seguridad firme hay **tres leyes principales** que todo usuario debería cumplir y que vamos a implementar en nuestro proyecto.

3.1 Ley 1: Backup periódico

Las **copias de seguridad** deben realizarse de forma periódica y llevándolas al día. La periodicidad con la que el usuario debe llevar a cabo las copias de seguridad depende de su situación y la frecuencia con la que trabaje con sus archivos.

3.2 Ley 2: Tener una copia de seguridad fuera y en el sitio

Tener más de una copia es fundamental para aumentar la seguridad. El usuario debería tener un mínimo de tres copias: la copia de trabajo interna, una copia de seguridad local y una copia de seguridad externa.

- La copia de seguridad local puede realizarse en un disco duro secundario o en uno externo.
- La copia de seguridad externa podemos almacenarla en la nube.

3.3 Ley 3: Automatización

Hay que **intentar automatizar** todos los archivos posibles. Cuanto más invisible sea el proceso de sus copias de seguridad mucho mejor.

Los servicios en la nube se encargan de la automatización de la copia de seguridad externa y muchos programas de copia de seguridad locales tienen una función similar o un programador para decidir qué hora del día para ejecutar la copia de seguridad local.

4. COPIAS DE SEGURIDAD DESDE EL PROPIO PROGRAMA

4.1 Respaldo de Base de Datos

La **copia de seguridad o backup** se refiere a la copia de archivos físicos o bases de datos a un sitio secundario para su preservación en caso de falla del equipo u otro error. El proceso de copia de seguridad de los datos es esencial para un plan de recuperación de datos.

Una copia de seguridad puede emplearse para reconstruir la base de datos. Es posible dividir las entre **copias físicas y copias lógicas**. El objetivo principal es garantizar una recuperación de datos rápida y confiable en caso de que resulte necesario. El proceso de recuperación de archivos de datos respaldados se conoce como restauración de archivos.

Para **hacer el backup** se podría guardar el archivo mediante un comando, que lo guardaría en un formato no ejecutable por ningún otro programa salvo el creado. Un ejemplo es “.bkp” o “.bckp”.

```
mysqldump -u "usuario" -p"contraseña" nombre-de-la-base-de-datos >
ubicación-y-nombre-del-respaldo.sql
```

4.2 Restauración de Base de Datos

Una **restauración de base de datos o restore** es la carga de una copia de seguridad al sistema, ya sea por pérdida de información, eliminación accidental u otras causas.

Para **restaurar un respaldo de una base de datos MySQL** usamos el siguiente comando:

```
mysql -u "usuario" -p"contraseña" nombre-de-la-base-de-datos <
ubicación-y-nombre-del-respaldo.sql
```

4.3 Respaldo y Restauración de MYSQL en Java

Esto es un ejemplo de cómo realizamos un **backup de una base de datos MySQL con Java** y cómo restaurarlo después:

4.3.1 Respaldo Base de datos de MySQL en Java

```
Process p = Runtime.getRuntime().exec("mysqldump -u prueba -p1234 tienda");
InputStream is = p.getInputStream();//Pedimos la entrada
FileOutputStream fos = new FileOutputStream("backup tienda.sql"); //creamos el
archivo para el respaldo
byte[] buffer = new byte[1000]; //Creamos una variable de tipo byte para el buffer
int leido = is.read(buffer); //Devuelve el número de bytes leídos o -1 si se alcanzó el final
del stream.
while (leido > 0) {
    fos.write(buffer, 0, leido);//Buffer de caracteres, Desplazamiento de partida para
empezar a escribir caracteres, Número de caracteres para escribir leido = is.read(buffer);
}
fos.close();//Cierra respaldo
```

4.3.2 Restaurar Base de datos de MySQL en Java

```
Process p = Runtime.getRuntime().exec("mysql -u prueba -p1234 database");

OutputStream os = p.getOutputStream();//Pedimos la salida
FileInputStream fis = new FileInputStream("backup pruebas.sql"); //creamos el archivo
para el respaldo
byte[] buffer = new byte[1000]; //Creamos una variable de tipo byte para el buffer

int leido = fis.read(buffer);//Devuelve el número de bytes leídos o -1 si se alcanzó el final
del stream.
while (leido > 0) {
    os.write(buffer, 0, leido); //Buffer de caracteres, Desplazamiento de partida para
empezar a escribir caracteres, Número de caracteres para escribir
    leido = fis.read(buffer);
}
```

```
os.flush(); //vacía los buffers de salida
os.close(); //Cierra
fis.close(); //Cierra respaldo
```

4.3.3 Proceso para mostrar en pantalla los errores que envía MySQL

```
public class HiloLector extends Thread {
    private final InputStream is;
    public HiloLector(InputStream is) {
        this.is = is;
    }

    @Override
    public void run() {
        try {
            byte[] buffer = new byte[1000];
            int leido = is.read(buffer);
            while (leido > 0) {
                String texto = new String(buffer, 0, leido);
                System.out.print(texto);
                leido = is.read(buffer);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

En el respaldo y restauración, sólo tenemos que poner esta línea después de las llamadas a `Runtime.getRuntime().exec(«...»):`

```
new HiloLector(p.getErrorStream()).start();
```

5. ENCAPSULACIÓN DE NUESTRA APLICACIÓN

5.1 Definición de Encapsulación Java

Nuestro proyecto hará uso de la **encapsulación en Java** consistente en la ocultación del estado o de los datos miembros de un objeto, de forma que sólo es posible modificar los mismos mediante los métodos definidos para dicho objeto.

Orientado a nuestra aplicación, el cliente podrá realizar varias apuestas para el mismo partido.

5.1.1 Datos

En los datos tendremos en cuenta lo siguiente:

- **Campos de Clase.** Los campos de la clase pueden ser establecidos como solo lectura o solo escritura.
- **Métodos.** Proveer métodos públicos para modificar y obtener los datos de la clase; en Java se utilizarían métodos con función de “getter” (para obtener datos) y “setters” (normalmente para cambiarle el valor a una variable).
- **Control.** Esto hace que la clase tenga un control total sobre los datos almacenados en sus campos.

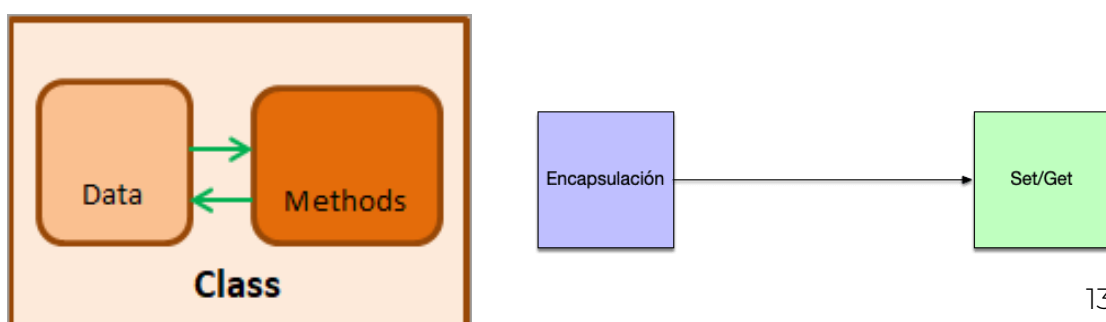


Figura 1

Figura 2

6. INSTALACIÓN

La **base de datos está protegida** por contener información sensible y no es accesible ni descargable por medios legales.

7. PROCEDIMIENTOS, FUNCIONES Y DISPARADORES

7.1 Procedimiento Almacenado

Un **procedimiento almacenado** es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales pueden en su lugar referirse al procedimiento almacenado. Se invocan lanzando el comando CALL.

7.2 Funciones Almacenadas

Una **función** es idéntica a un procedimiento pero, a diferencia de éste, pueden llamarse desde dentro de una línea de comando (el procedimiento requiere del comando CALL, únicamente).

7.3 Disparadores

Los **triggers o disparadores** son objetos de la base de datos que ejecutan acciones cuando se producen ciertos eventos (inserciones, modificaciones, borrados, creación de tablas, etc).

8. ESTRUCTURA

Vamos a estructurar nuestra base de datos mediante la **creación de tablas** que contienen datos referentes a las apuestas online de nuestra aplicación. Las tablas establecen relaciones entre ellas.

8.1 Creación de Tablas

8.1.1 La tabla de Usuarios

En la **tabla de clientes** le hemos asignado un ID a cada cliente. Esta tabla lista la información de todos los clientes contenidos en nuestra base de datos que se registran para realizar apuestas online en nuestra aplicación.

- **Usuario_id.** Clave primaria usada para identificar cada usuario en la tabla.
- **Nombre.** Nombre del cliente.
- **Apellidos.** Apellidos del cliente.
- **Cuenta Bancaria (Información tarjeta).** Datos bancarios del usuario. (encriptado)
- **Teléfono.** Teléfono del cliente.
- **E-mail.** E-mail del cliente.

- **País.** País del cliente.

8.1.2 La tabla de Equipos

En la tabla de equipos se asigna un ID a cada equipo. Esta tabla lista la información de todos los equipos en nuestra base de datos que se registran para realizar apuestas online en nuestra aplicación.

- **Equipo_id. Clave** primaria usada para identificar cada equipo en la tabla.
- **Equipo.** Nombre del equipo.
- **Media.** Estadísticas de las victorias del equipo en base a los resultados de la temporada.

8.1.3 La tabla de Apuestas

En la **tabla de apuestas** se asigna un ID a cada apuesta. Esta tabla lista todas las apuestas realizadas por los usuarios.

- **Apuesta_id.** Clave primaria usada para identificar cada apuesta en la tabla.
- **Equipo.** Equipo por el que se ha realizado la apuesta.
- **Apuesta.** Cantidad de dinero apostada.
- **Usuario.** Usuario que ha realizado la apuesta.

8.1.4 La tabla de Previsión

En la **tabla de previsión de resultados** le hemos asignado un ID a cada previsión. Esta tabla lista información de todas las apuestas que se llevarán a cabo en nuestra aplicación en base a una previsión de resultados.

- **Previsión_id.** Clave primaria usada para identificar cada equipo en la tabla.
- **Fecha.** Fecha de los enfrentamientos entre partidos.
- **Equipos enfrentados.** Nombre de los equipos que juegan.
- **Local / visitante.** Clasificación local o visitante.
- **Estadística resultados local / visitante.** Media de acierto de acuerdo a la clasificación local o visitante.
- **Previsión.** Previsión de resultados de la casa.

8.1.5 Asignación de Variables

Hemos realizado la **asignación de variables** de acuerdo al tipo de dato introducido.

8.1.5.1 Tabla de Usuarios

- Usuario_id (int).
- Nombre (string).
- Apellidos (string).
- Datos Bancarios (string).
- Teléfono (string).
- Email (string).
- País (string).

8.1.5.2 Tabla de Equipos

- Equipo_id (int).
- Equipo (String).
- Media (double).

8.1.5.3 Tabla de Apuestas

- Apuesta_id (int).
- Equipo (string).
- Apuesta (double).
- Usuario (string).

8.1.5.4 Tabla de Previsión

- Previsión_id (int).
- Fecha (date).
- Equipos enfrentados (string).
- Local / visitante (string).
- Estadística resultados local / visitante (int).
- Previsión (int).

9. DISEÑO DE LA ESTRUCTURA

A continuación veremos un **diseño inicial de la estructura** en el que podemos observar cuatro tablas diferenciadas (apuestas, usuarios, previsión, equipos), expuestas anteriormente.

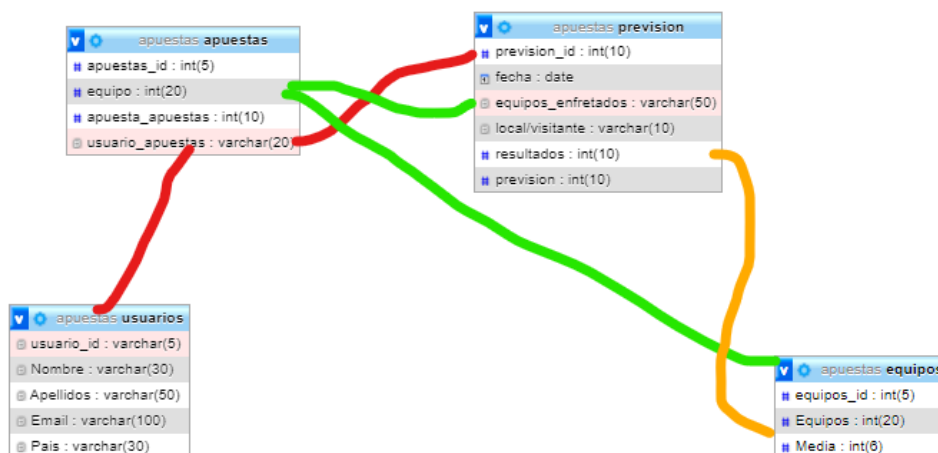


Figura 3

10. DISEÑO DE LA APLICACIÓN

10.1 Diseño de logo

Para el **logo de nuestra aplicación** hemos seleccionado las siguientes imágenes:



10.2 Diseño general de la Aplicación

Para el **diseño general de la aplicación** hemos hecho un primer boceto con algunos de los elementos iniciales que incluiremos.



10.3 Elementos de la Aplicación

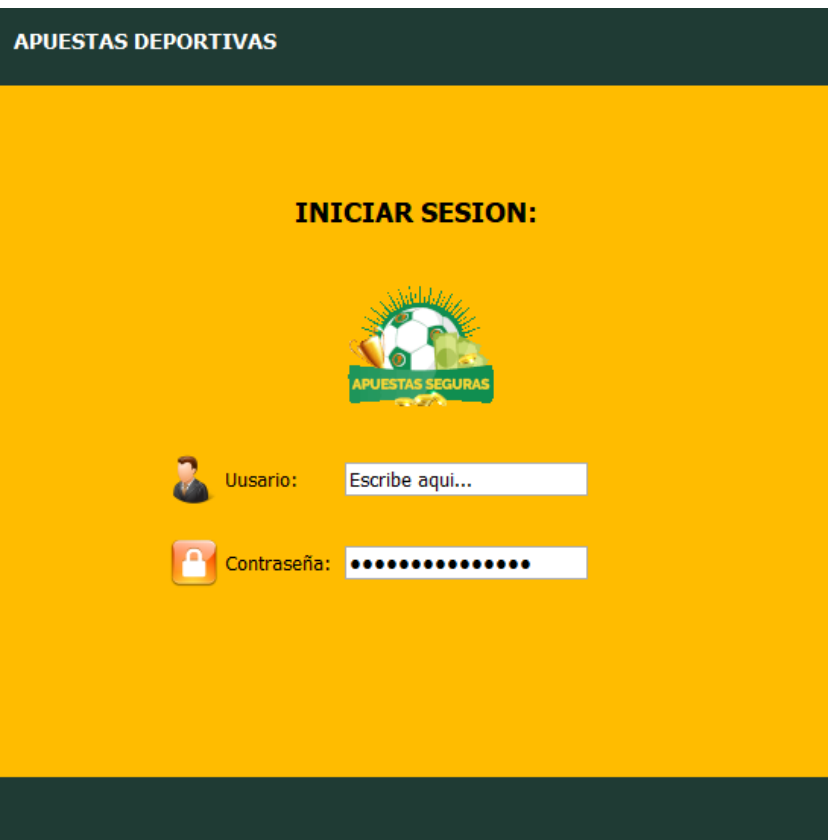
La aplicación contiene los siguientes **elementos**:

- Logo del programa, que incluimos más adelante.
- Barra de búsqueda de partidos, equipo y apuestas.
- Botón de inicio, que permite al usuario iniciar la aplicación de forma directa si ya ha habido un registro anterior.
- Botón de formulario de registro, que permite al usuario registrarse en la aplicación.
- Botón de ayuda, que permite al usuario obtener información sobre lo que realiza la aplicación.
- Botón de inicio de sesión, que incluye avatar de usuario.
- Pestañas, que incluirán diferentes secciones (calculadora de apuestas, wallet o cartera de información de las ganancias y apuestas actuales, estadísticas de las apuestas, ...).
- Tabla resumen, que permite al usuario obtener información sobre las apuestas que se realizan (fecha, liga, partido y apuesta).

Hemos realizado el diseño de tres elementos principales:

- Ventana de Inicio de Sesión
- Ventana de Carga
- Ventana Principal de la Aplicación

10.3.1 Ventana de Inicio de Sesión



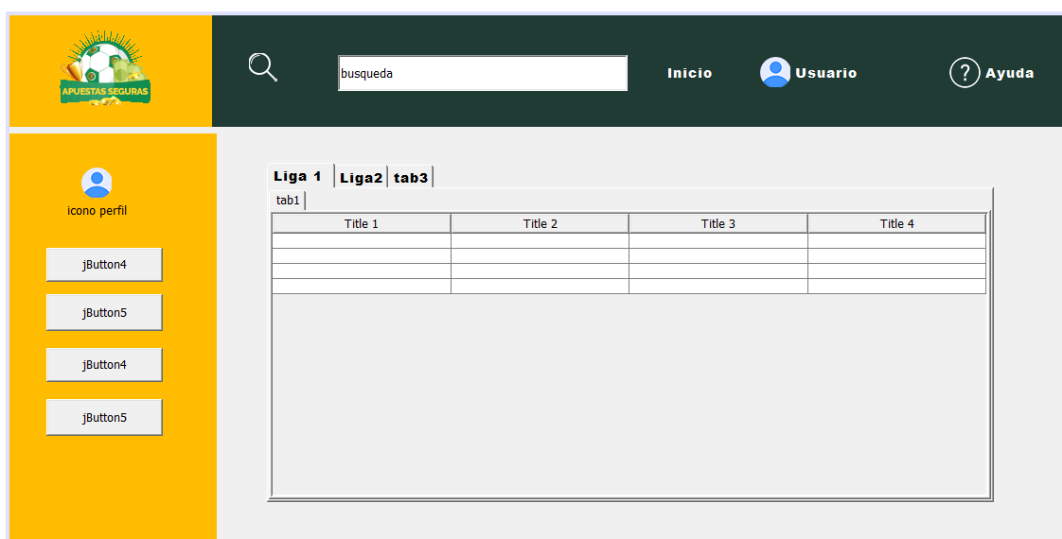
Boceto inicial de pantalla para iniciar sesión antes de arrancar la ventana principal de la aplicación con Usuario y contraseña.

10.3.2 Ventana de Carga



Boceto inicial de la pantalla de carga antes de arrancar el programa. La idea es que vaya apareciendo una barra de progreso con el porcentaje de carga.

10.3.3 Ventana Principal de la Aplicación



Boceto de entrada al programa una vez iniciada sesión en la aplicación.

10.3.4 Ventana Formulario

APUESTAS DEPORTIVAS

 **FORMULARIO DE REGISTRO:**

 **Nombre:**

 **Apellidos:**

 **Fecha de nacimiento:**

 **E-mail:**

 **Contraseña:**

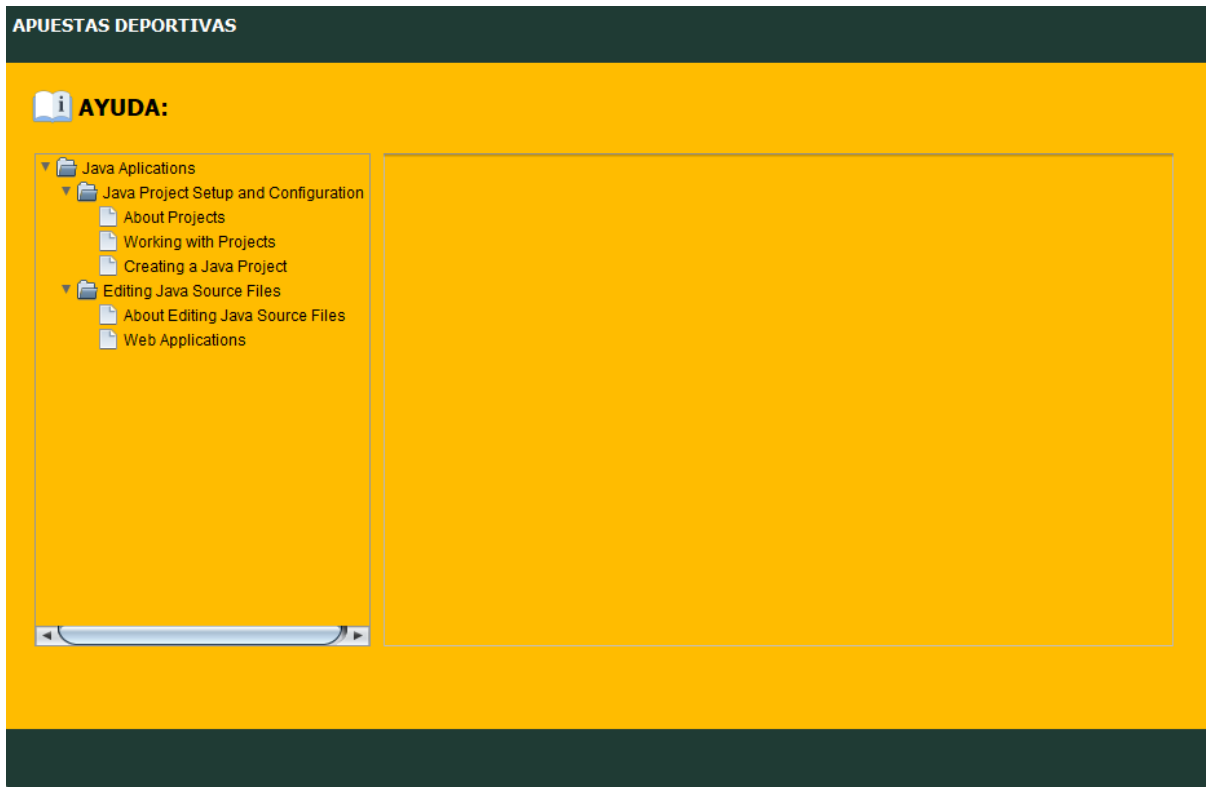
 **Verificar contraseña:**

 **¿ERES MAYOR DE 18 AÑOS?** ☐ SI ☐ NO

☐ Si eres mayor de 18 años: Aceptas las condiciones de uso y su política de privacidad

Boceto inicial de la pantalla registro de usuario al darle al botón que registro de la ventana principal de la aplicación.

10.3.5 Ventana Ayuda



Boceto inicial de la pantalla Ayuda, donde dará información de ayuda a los usuarios que lo necesiten y preguntas frecuentes hacia algún problema de nuestra aplicación.

10.4 Elementos que incluiremos en el futuro

En el futuro incluiremos:

- **Un conversor de monedas** y permitiremos al usuario visualizar las apuestas con la moneda elegida.
- **Una explicación del funcionamiento del programa**, así como un contacto con la organización para cualquier incidencia o duda adicional. Todo ello incluido en el botón de ayuda.

11. REDACCIÓN DE LA INFORMACIÓN

11.1 Información Sobre la Liga

Liga española de fútbol. Principal competición entre equipos de Fútbol de España. Se celebra anualmente desde 1929, y la temporada se desarrolla entre los meses de septiembre (o finales de agosto) y junio (o finales de mayo).

El torneo está integrado por un sistema piramidal de ligas (divisiones) interconectadas entre sí, cuya máxima categoría es la **Primera División** o LaLiga o por motivos de patrocinio como **LaLiga Santander**. Al término de cada temporada y en función de los resultados obtenidos, los equipos participantes pueden ascender o descender de división.

El sistema empleado en la **Liga Española de Fútbol** es piramidal, con una primera división a la que podrán acceder los equipos que vayan escalando posiciones desde la quinta categoría o divisiones regionales

Para poder participar en la Liga española los clubes deben estar federados. En el caso de las categorías profesionales (primera y segunda división), los clubes también deben cumplir los requisitos exigidos por la **Liga Nacional de Fútbol Profesional (LFP)**. A diferencia de otros países, los equipos filiales pueden participar en la Liga, aunque nunca en una división superior o igual a la del equipo al que pertenecen.

11.1.1 Funcionamiento de la Competición

Cada club se enfrenta dos ocasiones a los demás equipos de su misma división (una vez en campo propio y otra en el del contrario), siguiendo un calendario establecido por sorteo a principio de temporada. Generalmente se disputa una jornada cada semana (coincidiendo con el fin de semana).

Los encuentros finalizan a los noventa minutos, sin prórrogas ni tanda de penaltis en caso de terminar en empate.

En función del resultado al final de cada partido, los equipos obtienen una serie de puntos: tres para el ganador, cero para el perdedor y un punto para cada equipo en caso de empate.

Al término de la temporada se elabora una clasificación en función de los puntos acumulados a lo largo del campeonato. Si dos equipos terminaran en igualdad de puntos, el primer criterio de desempate es el resultado del enfrentamiento directo entre aquellos equipos que terminen el torneo en igualdad de puntos.

Segundo criterio: mejor diferencia entre goles anotados y recibidos, si el empate persiste, ganará el equipo que haya anotados más goles. En caso de que hayan sido tres o más los equipos implicados en el empate, se tiene en cuenta en primer lugar el número de puntos obtenidos en los enfrentamientos entre los clubes implicados, siguiendo por la diferencia de goles en los enfrentamientos entre dichos clubes.

Si no se hubiera acabado con la igualdad, se tendrían en cuenta los goles anotados y encajados durante toda la liga por los clubes empatados a puntos. El sistema es eliminatorio. Es decir, si con el primer método de desempate, un equipo desempata y persiste el empate entre otros dos, se volverá a aplicar todo el sistema a los clubes que sigan empatados, eliminando a los ya desempatados.

Todo esto ayuda a determinar al campeón de la liga regular al término de la última jornada. El equipo que suma más puntos al final del campeonato se proclama campeón de su categoría y, en el caso de la **Primera División**, el primer clasificado es considerado el campeón de Liga. Dado que se trata de un sistema jerárquico de ligas interconectadas entre sí, los primeros clasificados de cada división tienen la opción de ascender a la categoría superior para la siguiente temporada, sustituyendo a los últimos

clasificados de dicha categoría, que son degradados. El número de equipos que ascienden y descienden varía en función de cada categoría.

Los cuatro primeros clasificados de la primera división acceden a participar en la máxima competición europea de la siguiente temporada: la **Liga de Campeones**. Los tres primeros obtienen clasificación directa, mientras que el cuarto disputa una eliminatoria de clasificación (en caso de ser eliminado en esas eliminatorias, pasarán a participar en la **Europa League**, antigua Copa de la UEFA). Estas plazas son variables y dependen del coeficiente UEFA de España, referente a sus equipos, para que a la **Liga de Campeones** accedan los equipos más poderosos ya que los equipos españoles o italianos son más fuertes, por ejemplo, que los escoceses; por ello los dos primeros disponen de 4 plazas y los escoceses tan sólo de 1. Esta norma es para todas las competiciones europeas.

El quinto y sexto clasificado obtienen la clasificación directa para participar la siguiente temporada en la **Europa League**, junto al campeón de la **Copa del Rey**. Si éste está clasificado entre los puestos 5º y 6º (clasificado para la Europa League), el equipo que obtenga la séptima posición también juega la Europa League, caso que también se dará cuando los dos finalistas de la Copa del Rey se hubiesen clasificado para la Liga de Campeones. Si sólo el campeón se clasificó para la Liga de Campeones, participará en la Europa League el subcampeón de la Copa del Rey.

11.1.2 Reglamento de la Competición

La justicia deportiva de la competición está gestionada por el **Comité de Competición de la Liga**, organismo encargado de imponer sanciones y dirimir los conflictos entre clubes. Los árbitros (todos españoles) de los partidos son designados, a través de un comité de designación formado por tres miembros: Antonio Jesús López Nieto (ex-árbitro malagueño

internacional), Evaristo Puentes Leira (ex-árbitro gallego de 1ª división) y Victoriano Sánchez Arminio (presidente del Comité Técnico de Árbitros).

Un árbitro no puede dirigir un partido en el que participe un equipo de la misma comunidad regional en la que esté colegiado.

La **Real Federación Española de Fútbol** ejerce, por delegación de la Ley 10/1990, de 15 de octubre, la organización, en exclusiva, de las competiciones oficiales de ámbito estatal, y coordinadamente con la LFP las calificadas como profesionales. A tal fin se dicta un Reglamento General que regula diferentes aspectos de las competiciones federativas.

La última revisión del Reglamento General fue aprobada por la Comisión Delegada de la RFEF y posteriormente por la Comisión Directiva del Consejo Superior de Deportes.

El Reglamento General consta de 275 artículos.

La Real Federación Española de Fútbol ejerce por delegación de la UEFA la facultad de expedir licencias de participación de los clubes españoles en las competiciones europeas.

De acuerdo con lo dispuesto en el Reglamento todos los clubes españoles de fútbol de Primera División, y en su caso, cualquiera otro que por razones deportivas fueren a participar en cualesquiera de las competiciones UEFA, deberán solicitar a la RFEF la concesión de una licencia nacional especial "**la Licencia UEFA**" que habilite su participación en dichas competiciones.

El **Departamento de la Licencia UEFA** es el encargado de la gestión directa de esta licencia para todos los clubes, que se instrumentaliza y regula a través de este Reglamento.

Este nuevo reglamento viene a implementar, a nivel nacional, el contenido del nuevo **Reglamento FIFA** de Intermediarios, aprobado por el Comité Ejecutivo de la FIFA el 21 de marzo de 2014, y que sustituye al Reglamento

sobre los Agentes de Jugadores, enmendado por última vez el 29 de octubre de 2007. Mediante el mismo, se introdujeron sustanciales modificaciones en el sistema rector de agentes de jugadores, conllevando una importante reforma en el régimen jurídico aplicable a los mismos. En este sentido, el presente reglamento que ahora se pone a disposición de los afiliados a la RFEF, ha sido elaborado de conformidad con lo establecido en el artículo 1.2 del “Reglamento sobre las relaciones con intermediarios” de **FIFA**, que exige a las asociaciones nacionales implantar y aplicar, al menos, los principios, normas y requisitos mínimos establecidos en el mismo.

11.1.3 Estadísticas Últimas Temporadas

2000-2001	REAL MADRID	DEPORTIVO	MALLORCA
2001-2002	VALENCIA	DEPORTIVO	REAL MADRID
2002-2003	REAL MADRID	REAL SOCIEDAD	DEPORTIVO
2003-2004	VALENCIA	FC BARCELONA	DEPORTIVO
2004-2005	FC BARCELONA	REAL MADRID	VILLARREAL
2005-2006	FC BARCELONA	REAL MADRID	VALENCIA
2006-2007	REAL MADRID	FC BARCELONA	SEVILLA
2007-2008	REAL MADRID	VILLARREAL	FC BARCELONA
2008-2009	FC BARCELONA	REAL MADRID	SEVILLA
2009-2010	FC BARCELONA	REAL MADRID	VALENCIA
2010-2011	FC BARCELONA	REAL MADRID	VALENCIA
2011-2012	REAL MADRID	FC BARCELONA	VALENCIA
2012-2013	FC BARCELONA	REAL MADRID	ATLÉTICO MADRID
2013-2014	ATLÉTICO MADRID	FC BARCELONA	REAL MADRID

2014-2015	FC BARCELONA	REAL MADRID	ATLÉTICO MADRID
2015-2016	FC BARCELONA	REAL MADRID	ATLÉTICO MADRID
2016-2017	REAL MADRID	FC BARCELONA	ATLÉTICO MADRID
2017-2018	FC BARCELONA	ATLÉTICO MADRID	REAL MADRID
2018-2019	FC BARCELONA	ATLÉTICO MADRID	REAL MADRID
2019-2020	REAL MADRID	FC BARCELONA	ATLÉTICO MADRID
2020-2021	ATLÉTICO MADRID	REAL MADRID	FC BARCELONA

11.2 Información Sobre los Equipos

Los equipos que participan en La Liga 2021/22 son: Athletic Club de Bilbao, Atlético de Madrid, Barcelona, Betis, Cádiz, Celta, Elche, Granada, Levante, Mallorca, Osasuna, Rayo Vallecano, Real Madrid, Real Sociedad, Sevilla, Valencia y Villarreal.

12. COMPONENTES DE LA APLICACIÓN

A continuación realizaremos una breve descripción de los componentes, listados seguidamente, que contendrá nuestra aplicación.

12.1 Lista de Componentes (Imports)

- ❖ java.sql
- ❖ java.time
- ❖ java.util.logging
- ❖ java.awt.image
- ❖ java.rmi.registry
- ❖ java.awt.event

- ❖ java.swing
- ❖ javax.sql
- ❖ javax.swing
- ❖ javax.imageio
- ❖ java.io
- ❖ java.awt.print

12.2 Import JAVA.SQL*

Implementada por la API (Interfaz de Programación de Aplicaciones) de JDBC. Es una paquetería importada por Java para hacer consultas en Base de Datos (BD). Si se usa únicamente esta librería, entonces podemos realizar una conexión básica y simple desde consola.

Importa clases para hacerlas visibles

- Sirve para acceder a la base de datos SQL mediante una llamada a través del uso de la librería.
- Contiene su propia estructura y hace una llamada de sus propias clases. Lo primero que se importa son los paquetes o clases que se van a utilizar en la nueva clase.
- El asterisco (*) indica que todas las clases del paquete java.sql serán importadas.

12.3 Import JAVA.TIME

La API principal para fechas, horas, instantes y duraciones.

Incluir temporalidad

Java.time incluye muchas clases, pero las básicas son:

- **LocalDate**: representa a fechas sin la hora y nos facilita su manejo para declararlas, sumar y restar fechas y compararlas.
- **LocalTime**: es idéntica a la anterior pero para el manejo de horas, sin ninguna fecha asociada, pudiendo así compararlas, sumar o restar tiempo a las mismas...
- **LocalDateTime**: como puedes suponer, es una combinación de las dos anteriores, que permite hacer lo mismo con fechas y horas simultáneamente.
- **Instant**: es muy parecida a la anterior pero a la vez muy diferente. Se usa para almacenar un punto determinado en el tiempo, o sea con fecha y hora, pero guarda su valor como un *timestamp* de UNIX, es decir, en nanosegundos desde el *epoch* de UNIX (1/1/1970 a las 00:00) y usando la zona horaria UTC. Es muy útil para manejar momentos en el tiempo de manera neutra e intercambiarlo entre aplicaciones y sistemas, por lo que lo verás utilizado muy a menudo.
- **ZonedDateTime**: esta clase es como la `LocalDateTime` pero teniendo en cuenta una zona horaria concreta, ya que las anteriores no la tienen en cuenta.
- **Period**: esta clase auxiliar nos ayuda a obtener diferencias entre fechas en distintos periodos (segundos, minutos, días...) y también a añadir esas diferencias a las fechas.
- **Duration**: esta es muy parecida a la anterior pero para manejo de horas exclusivamente.

12.4 Import JAVA.UTIL.LOGGING

Cuando se está preparando un programa para un entorno de producción tener un log donde se reporten los eventos y errores puede ser la diferencia entre pasar una semana tratando de replicar un error o solo leer un archivo y saber en que linea ocurrió el error y si bien hay todo un mundo de librerías y frameworks para este propósito no hay que olvidar

que el propio Java ya contiene las clases para hacer esto y que nunca esta de mas ahorrarse dependencias.

Realización de un registro de eventos

La forma en que opera el framework de logging de Java es la siguiente:

1. Creamos un objeto estático de tipo **Logger** desde el cual enviaremos los mensajes a registrar en la bitácora
2. Creamos un objeto **ConsoleHandler** y se lo agregamos al **Logger**, de modo que los mensajes aparezcan automáticamente en la consola
3. Creamos un objeto **FileHandler** y se lo agregamos al **Logger**, este Handler en particular enviará los mensajes al archivo que le indiquemos.
4. Creamos un objeto **SimpleFormatter** y lo establecemos en el **FileHandler**, de este modo los logs se escriben como texto plano simple, de no indicarlo se escribirán en formato XML por defecto
5. Para registrar algo en bitácora llamamos al método **log** del **Logger** indicamos el nivel del log y el mensaje que deseamos registrar, esto automáticamente reportará la fecha, hora, el nombre completo de la clase y el método y en el caso de mensajes de nivel grave la línea de código donde se genero el reporte

Niveles de log

En nuestro log necesitamos indicar el “nivel” de ese evento, estos nos permiten diferenciar entre meras notificaciones de que todo está bien a errores de diferente severidad esto no solo se refleja en el archivo sino que también podremos filtrar a partir de qué nivel de error queremos que se registre de modo que no se llene el archivo de meras notificaciones, este filtrado se indica en el objeto Handler.

12.5 Import JAVA.AWT.IMAGE

Java permite incorporar imágenes de tipo GIF y JPEG definidas en ficheros. Se dispone para ello de la **clase java.awt.Image**.

Importar clases para incorporar imágenes

Para cargar una imagen hay que indicar la localización del archivo y cargarlo mediante el **método getImage()**. Este método existe en las clases **java.awt.Toolkit**.

Entonces, para cargar una imagen hay que comenzar creando un objeto (o una referencia) **Image** y llamar al método **getImage() (de Toolkit)**; Una vez cargada la imagen, hay que representarla, para lo cual se redefine el **método paint()** para llamar al método **drawImage()** de la clase Graphics.

Los **objetos Graphics** pueden mostrar imágenes a través del método: **drawImage()**. Dicho método admite varias formas, aunque casi siempre hay que incluir el nombre del objeto imagen creado.

Clase Image

Una imagen es un objeto gráfico rectangular compuesto por pixels coloreados. Cada pixel en una imagen describe un color de una particular localización de la imagen.

12.6 Import JAVA.RMI.REGISTRY

Permite a una aplicación buscar objetos que están siendo exportados para su uso mediante llamadas a métodos remotos.

Búsqueda de objetos con llamadas a métodos

Una vez que el objeto ha sido localizado, ya se puede utilizar utilizando la misma sintaxis que una llamada a un método local.

Para encontrar los objetos, **RMI** utiliza un servicio que mantiene una tabla de direcciones de objetos remotos que están siendo exportados por sus aplicaciones.

A todos los objetos se les asigna nombres únicos que se utilizan para identificarlos. Algunos métodos pueden llamarse desde la **interfaz `rmi.registry.Registry`**, o desde la **clase `rmi.Naming`**, que permite añadir, eliminar y acceder a objetos remotos en la tabla de registro de objetos.

El servidor del servicio de nombres registra los objetos mediante **llamadas a `bind()` o `rebind()`** sobre una instancia de un registro del objeto que está siendo exportado. Ya que todos los objetos tienen nombres únicos, una llamada a `bind()` a un registro que contiene un nombre (String) que ya está registrado provocará una excepción. De forma alternativa, `rebind()` reemplaza un objeto antiguo con un nombre dado, con un nuevo objeto.

12.7 Import JAVA.AWT.EVENT

Para controlar los eventos en nuestras aplicaciones de Java creadas con **interfaz gráfica**, como detectar las pulsaciones y movimientos del ratón y

pulsaciones del teclado, es necesario utilizar el paquete de **AWT** llamado **Event**.

Control de eventos en Java

Con esta librería, registramos lo dicho anteriormente mediante eventos, que se diferencian entre:

- Eventos de teclado: Mediante eventos de teclado o **KeyEvent**, el programa puede detectar qué teclas pulsa el usuario y realizar una acción cuando la pulsa y cuando suelta ésta.
- Eventos de ratón: Parecido a los eventos de teclado, **Event** captura los eventos de ratón o **MouseEvent**. Detecta qué botón del ratón pulsas, si arrastras el ratón mientras clicas, generar una acción cuando pulsas el botón y también cuando lo sueltas.

12.8 Import JAVA.SWING

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como: cajas de texto, botones, listas desplegables y tablas.

Incluye elementos para interfaz gráfica

- El diseño en Java puro posee menos limitaciones de plataforma.
- El desarrollo de componentes Swing es más activo.
- Los componentes de Swing soportan más características.

12.9 Import JAVA.IMAGE.IO

Es una clase de utilidad que proporciona muchos métodos de utilidad relacionados con el procesamiento de imágenes en Java.

El más común de ellos es leer el archivo de imagen del formulario y escribir imágenes en el archivo en Java. Puede escribir cualquiera de las imágenes **.jpg, .png, .bmp** o .gif para archivar en Java. Al igual que escribir, la lectura también es fluida con ImageIO y puede leer BufferedImage directamente desde la URL.

La lectura de imágenes es un poco diferente a la lectura de texto o archivos binarios en Java , ya que están asociados con un formato diferente. Aunque aún puede usar el enfoque **getClass().getResourceAsStream()** para cargar imágenes.

Además del soporte aparente de **lectura y escritura de imágenes en Java** , la clase ImageIO también contiene muchos otros métodos de utilidad para ubicar **ImageReaders** e **ImageWriters** y realizar la codificación y decodificación.

Eso es todo en este **archivo de imagen de lectura** y escritura en Java usando javax.imageio.ImageIO . Avíseme si tiene algún problema al probar estos ejemplos de ImageIO.

12.10 Import JAVA.IO

El paquete java.io es el encargado de gestionar las **operaciones de entrada/salida**.

Operaciones de entrada y salida

Entrada estándar sería **System.in** (es un objeto **InputStream**), salida estándar sería **System.out** y salida estándar de errores sería **System.err** (las salidas son objetos **PrintStreams**).

Cualquier programa realizado en Java que necesite llevar a cabo una operación de **I/O** lo hará a través de un **stream**.

Un **stream**, cuya traducción literal es «flujo», es una abstracción de todo aquello que produzca/escriba o consuma/lea información/bytes.

12.11 Import JAVA.AWT.PRINT

Proporciona clases e interfaces para una **API de impresión general**.

Impresión general

La API incluye características tales como:

- La capacidad de especificar tipos de documentos.
- Mecanismos para el control de configuración de página y formatos de página.
- La capacidad de gestionar los diálogos de control de trabajos.

13. LENGUAJE HTML Y CONEXIONES CON EL SERVIDOR

Lenguaje HTML

HTML es el lenguaje de marcado estándar para las páginas web. Con HTML puedes crear tu propio sitio web.

Apache Tomcat

Apache Tomcat es un contenedor Java Servlet, o contenedor web, que proporciona la funcionalidad extendida para interactuar con Java Servlets, al tiempo que implementa varias especificaciones técnicas de la plataforma Java: JavaServer Pages (JSP), Java Expression Language (Java EL) y WebSocket.

Servlet de Java

Este es un software que permite que un servidor web maneje contenido web dinámico **basado en Java utilizando el protocolo HTTP**.

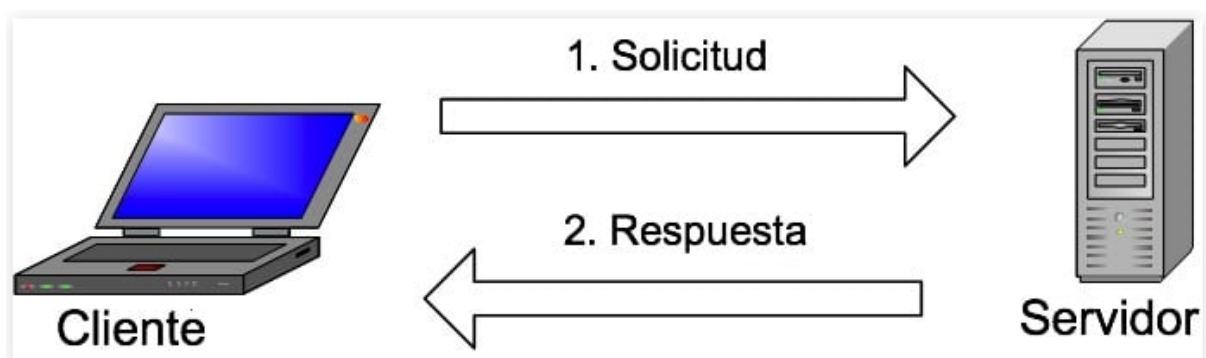
JSP es una tecnología similar que permite a los desarrolladores crear contenido dinámico utilizando documentos HTML o XML. En términos de su capacidad para habilitar contenido dinámico, los **Servlets de Java y JSP** son ampliamente comparables a PHP o ASP.NET, solo basados en el lenguaje de programación Java.

Al reunir todas estas tecnologías basadas en Java, Tomcat ofrece un entorno de servidor web «Java puro» para ejecutar aplicaciones creadas en el lenguaje de programación Java.

MYSQL

MySQL es un sistema de gestión de bases de datos relacionales de código abierto (RDBMS, por sus siglas en inglés) con un modelo cliente-servidor. RDBMS es un software o servicio utilizado para crear y administrar bases de datos basadas en un modelo relacional.

¿Cómo funciona MYSQL?



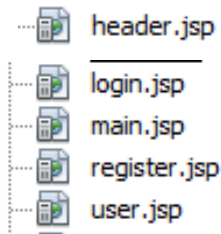
La imagen explica la **estructura básica cliente-servidor**. Uno o más dispositivos (clientes) se conectan a un servidor a través de una red específica. Cada cliente puede realizar una solicitud desde la interfaz gráfica de usuario (GUI) en sus pantallas, y el servidor producirá el output deseado, siempre que ambas partes entiendan la instrucción.

Los procesos principales que tienen lugar en un entorno MySQL son :

- **MySQL crea una base de datos** para almacenar y manipular datos, definiendo la relación de cada tabla.
- Los clientes pueden **realizar solicitudes** escribiendo instrucciones SQL específicas en MySQL.
- **La aplicación del servidor** responderá con la información solicitada y esta aparecerá frente a los clientes.

14. Java Servers Pages (JSP)

A continuación expondremos las vistas que se han implementado en la creación de la web.



Main

Se ha creado la vista principal “**main.jsp**”. Es un elemento que permite que se pueda ejecutar un programa.

Header

Se ha creado la vista de apoyo “**header.jsp**”, que se importa en las vistas que lo precisen para tener este código centralizado. Hace referencia a la barra superior de la página web donde se muestran tres botones (inicio, registro y ayuda).

Login

Se ha creado la vista “**login.jsp**” que hace referencia a la pestaña de inicio de sesión del usuario.

Register

Se ha creado la vista “**register.jsp**” que hace referencia a la pestaña de registro de usuario.

15. Conexión Base de Datos

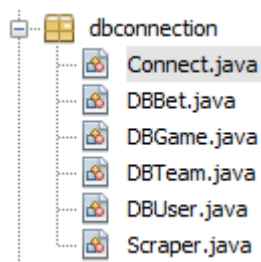
Repositorio o Manager

Un **repositorio de Installation Manager** es una carpeta de archivos con estructura de árbol que incluye la carga útil del producto y los metadatos.

Es un lugar centralizado donde se almacena información digital, normalmente **base de datos o archivos informáticos**. De este modo las clases creadas se comunican con una tabla de la base de datos y mapean esa información en la clase modelo correspondiente.

Base de Datos en Java

Se han generado una serie de clases dentro de un paquete llamado “**dbconnection**” que conforman la base de datos a utilizar en la página web.



Clase Connect (connect.java)

Clase principal del repositorio que establece la **conexión entre las diferentes clases y la información** que deberá estar contenida en ellas. Es un manejador de conexiones hacia un modelo de base de datos.

Clase DBBet (DBBet.java)

Clase secundaria que recoge **información sobre las apuestas** realizadas por el usuario.

Clase DBGame (DBGame.java)

Clase secundaria que recoge **información sobre los partidos** que tendrán lugar.

Clase DBTeam (DBTeam.java)

Clase secundaria que recoge **información sobre los equipos** que se enfrentarán.

Clase DBUser (DBUser.java)

Clase secundaria que recoge **información del registro de los usuarios** de la web.

Clase Scraper (Scraper.java)

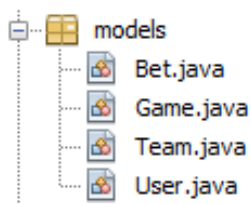
Clase complementaria utilizada para **actualizar la base de datos**. Obtiene de forma automática datos de una o varias páginas web.

16. Clases Modelo en Java

Las **clases modelo en java** conforman un conjunto de instrucciones para construir un tipo específico de objeto. Determina cómo se comportará un objeto y qué contendrá.

Este tipo de clases **representan las tablas** y se utiliza a modo de empaquetar la información que será intercambiada entre los manager, servlets y listas.

Se han generado una serie de clases dentro del paquete “**models**” que se expondrán seguidamente.



Clase Bet (Bet.java)

Clase que contiene información sobre las **apuestas**.

Clase Game (Game.java)

Clase que contiene información sobre los **partidos**.

Clase Team (Team.java)

Clase que contiene información sobre los **equipos**.

Clase User (User.java)

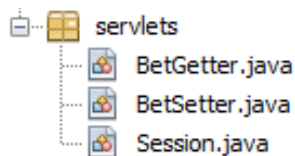
Clase que contiene información sobre los **usuarios**.

17. Clases Servlets en Java

Un **servlet** es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor.

El Servlet, que es una objeto java, se encarga de **generar el texto de la página web** que se entrega al contenedor. Median las peticiones entre la vista y los managers.

Dentro del paquete “**servlets**” se han generado una serie de clases.



Clase BetGetter (BetGetter.java)

Clase que nos sirve para **obtener, recuperar o acceder**, al valor ya asignado a un atributo y utilizarlo. En este caso trabajamos con información de apuestas.

Clase BetSetter (BetSetter.java)

Clase que nos sirve para **asignar un valor inicial** a un atributo, nos permite dar acceso a ciertos atributos que deseamos que el usuario pueda modificar. En este caso trabajamos con información de apuestas.

Clase Session (Session.java)

Clase que nos sirve para que el usuario realice el **registro o inicio de sesión**.

- **ayuda**
- **info pagos**
- **iniciar**
- **main**
- **login**
- **registro**

18. NORMAS DE PROGRAMACIÓN

Reutilizar código

Para **reutilizar el código** creamos nuevas clases pero, en lugar de partir de cero, partimos de clases, relacionadas con nuestra clase, que han sido ya creadas y depuradas. El truco está en usar las clases sin ensuciar el código existente.

Una forma de hacer esto es crear objetos de nuestras clases existentes dentro de la nueva clase. Esto se conoce como composición porque la nueva clase está compuesta de objetos de clases existentes. Estamos reutilizando la funcionalidad del código, y no la forma.

Otra forma es crear una nueva clase como un tipo de una clase ya existente. Tomamos la forma de la clase existente y añadimos código a la nueva, sin modificar la clase existente. Esta forma de crear nuevos objetos se llama herencia, y lo que hacemos es extender la clase en la que nos basamos para crear la nueva.

Escribir código legible y mantenible

Se considera que el **código es limpio** cuando es fácil de leer y entender. Si resuelve los problemas sin agregar complejidad innecesaria, permitiendo que el mantenimiento o las adaptaciones, por algún cambio de requerimiento, sean tareas más sencillas, entonces estamos hablando de “clean code”.

Para crear código limpio hay que conocer y poner en práctica un conjunto de principios o técnicas de desarrollo que nos ayudarán a evitar los *code smells*, es decir, esos síntomas de un programa que te dan el indicio de que existe un problema más profundo.

Estándares de codificación

Estas normas son muy útiles por muchas razones, entre las que destacan:

- Facilitan el mantenimiento de una aplicación. Dicho mantenimiento constituye el 80% del coste del ciclo de vida de la aplicación.
- Permite que cualquier programador entienda y pueda mantener la aplicación. En muy raras ocasiones una misma aplicación es mantenida por su autor original.
- Los estándares de programación mejoran la legibilidad del código, al mismo tiempo que permiten su comprensión rápida.

Definir convenciones de nomenclatura

Es un **conjunto de reglas** para la elección de la secuencia de caracteres que se utilice para los identificadores que denoten variables, tipos, funciones y otras entidades en el código fuente y la documentación.

Algunas de las razones para utilizar una convención de nombres (en lugar de permitir a los programadores elegir cualquier secuencia de caracteres) son:

- Reducir el esfuerzo necesario para leer y entender el código fuente.
- Mejorar la apariencia del código fuente, por ejemplo, al no permitir nombres excesivamente largos o abreviaturas poco claras.

La elección de las convenciones de nombres puede ser un problema de enorme polémica, donde los partidarios de cada convención consideran la suya como la mejor y las demás inferiores; coloquialmente, se dice que es una cuestión de dogma. Muchas empresas también han establecido su propio conjunto de convenciones para satisfacer mejor sus intereses.

Comentar el código

Los **comentarios en Java** y en cualquier lenguaje de programación son una herramienta que sirve para apoyar la documentación de los programas que desarrollamos y así facilitar su posterior comprensión por parte de alguna otra persona que comprenda algo de Java o el lenguaje en particular. Los comentarios, son líneas de código, que no son tenidas en cuenta por el compilador en el momento de ejecutar nuestra aplicación (es como si no estuviesen allí), por lo tanto no están sujetas a restricciones de sintaxis ni nada similar y podremos escribir cualquier cosa en éstas. El uso principal de las líneas de comentario en Java, es dar orden al código y hacerlo más comprensible, en especial cuando serán terceros los que verán y deberán entender nuestro programa (como dije anteriormente). Por ejemplo es muy común usar las líneas de comentarios, para dar una breve

explicación de cómo funciona cierta parte de un código, lo cual permite identificar todo con mayor rapidez.

Sangrado

El sangrado (también conocido como indentación) deberá aplicarse a toda estructura que esté lógicamente contenida dentro de otra. El sangrado será de un tabulador. Es suficiente entre 2 y 4 espacios. Para alguien que empieza a programar suele ser preferible unos 4 espacios, ya que se ve todo más claro.

Las líneas no tendrán en ningún caso demasiados caracteres que impidan que se pueda leer en una pantalla. Un número máximo recomendable suele estar entre unos 70 y 90 caracteres, incluyendo los espacios de sangrado. Si una línea debe ocupar más caracteres, tiene que dividirse en dos o más líneas. Para dividir una línea en varias, utiliza los siguientes principios:

- Tras una coma.
- Antes de un operador, que pasará a la línea siguiente.
- Una construcción de alto nivel (por ejemplo, una expresión con paréntesis).
- La nueva línea deberá alinearse con un sangrado lógico, respecto al punto de ruptura

Dividir el código

Cada método debe poseer una funcionalidad clara y simple, por lo que debemos separar los métodos que cambian de estado de aquellos que los

consultan. De esta manera, simplificamos el control de concurrencia y extensiones por herencia.

Utilizar una estructura de directorios

La estructura es similar a la de los directorios y ficheros. De hecho, puedes entender el nombre *package* como sinónimo de *directorio*. Los ficheros hacen el papel de las clases Java y los directorios hacen el papel de paquetes. La estructura de directorios en la que se organizan los ficheros .class (estructura física del sistema operativo) debe corresponderse con la estructura de paquetes definida en los ficheros fuente .java.

Documentar y compartir funciones internas

Documentar un proyecto es algo fundamental de cara a su futuro mantenimiento. Cuando programamos una clase, debemos generar documentación lo suficientemente detallada sobre ella como para que otros programadores sean capaces de usarla sólo con su interfaz. No debe existir necesidad de leer o estudiar su implementación, lo mismo que nosotros para usar una clase del API Java no leemos ni estudiamos su código fuente.

Javadoc es una utilidad de Oracle para la generación de documentación de APIs en formato HTML a partir de código fuente Java. Javadoc es el estándar para documentar clases de Java. La mayoría de los IDEs utilizan javadoc para generar de forma automática documentación de clases. BlueJ también utiliza javadoc y permite la generación automática de documentación, y visionarla bien de forma completa para todas las clases

de un proyecto, o bien de forma particular para una de las clases de un proyecto.

Veamos en primer lugar qué se debe incluir al documentar una clase:

- Nombre de la clase, descripción general, número de versión, nombre de autores.
- Documentación de cada constructor o método (especialmente los públicos) incluyendo: nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general, descripción de parámetros (si los hay), descripción del valor que devuelve.

Implementar el control de versiones

La característica principal en que se dividen bien podría ser si se trata de un sistema centralizado o no. En el primer caso, existe un servidor común donde se encuentra el código fuente, tanto la versión actual en desarrollo como todas aquellas versiones intermedias desde que dio comienzo el proyecto. En el último caso, no es necesario (aunque a menudo es recomendable) poseer un servidor común, sino que se pueden enviar y recibir actualizaciones de cada uno de los miembros participantes de forma directa.

¿Cómo funciona un sistema de control de versiones?

Lo habitual es que cada programador realice los cambios necesarios en el código fuente para la tarea que se le ha encomendado. Una vez que dichos cambios están listos, los envía al servidor (o a los otros participantes), de modo que el resto pueda recibirlos en cualquier momento, y así trabajar

sobre dichos cambios cuando tengan que realizar cualquier otra tarea. Se puede dar el caso de que varios programadores trabajen sobre el mismo fichero o ficheros, en cuyo caso el sistema lo detectará, y actuará para evitar posibles conflictos. Se pueden dar dos casos:

- Los programadores han trabajado en porciones de código diferentes: En principio, no se han pisado las líneas en las que han trabajado, así que es probable que sea suficiente efectuar ambos cambios sobre el fichero, sin más. Casi todos los sistemas de control de versiones detectan esta situación y realiza la unión de los cambios de forma automática, notificándolo al usuario para que tenga constancia.
- Los programadores han trabajado en líneas de código comunes, modificando, eliminando o añadiendo líneas en la misma porción de código: En estos casos, el sistema suele señalar que hay un conflicto entre ambos cambios, y habitualmente genera un fichero intermedio convenientemente marcado para que se puedan revisar ambos cambios de forma simultánea, y así quedarse con uno, con el otro, o con una combinación de los dos, realizando la unión a mano y eliminando lo que sobra.

Seleccionar un entorno de desarrollo

Lo primero que debemos hacer a la hora de elegir un **entorno de desarrollo** o IDE es tener claro los lenguajes de programación que vamos a utilizar. Como estamos hablando de desarrollo web tenemos claro que si o si vamos a utilizar HTML, CSS y JavaScript (Frontend). En la parte de servidor (Backend) tenemos varias opciones, las más comunes son PHP, Java y ASP.NET. Dependiendo del Backend elegiremos un IDE u otro.

Cuando estamos comparando los IDEs que nos ofrece el mercado debemos plantearnos ciertas cuestiones:

- Coloreado de sintaxis para una mejor legibilidad.
- Que permita insertar trozos de código o snippets.
- Integración con un sistema de control de versiones.
- Poder crear proyectos a partir de plantillas o templates.
- Función de autocompletado de código.
- Ejecución en modo debug.
- Buscar y reemplazar código.
- Refactorizar código.

Prototipos

Es un **patrón de diseño** creacional que permite la clonación de objetos, incluso los complejos, sin acoplarse a sus clases específicas.

Todas las clases prototipo deben tener una interfaz común que haga posible copiar objetos incluso si sus clases concretas son desconocidas. Los objetos prototipo pueden producir copias completas, ya que los objetos de la misma clase pueden acceder a los campos privados de los demás.

Separar lógica y contenido

La **programación por capas** es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Utilizar optimización

Para conseguir optimizar el rendimiento, puede llevar a cabo estas acciones:

- Mejore el rendimiento del código Java utilizando el compilador **Just-In-Time** (JIT) o utilizando una caché de clase compartida.
- Establezca cuidadosamente los valores para obtener un rendimiento de recogida de basura óptimo.
- Solo debe utilizar los métodos nativos para iniciar funciones del sistema que sean de relativamente larga ejecución y que no estén disponibles directamente en Java.
- Utilice excepciones Java en los casos en que no se produzca el flujo normal por la aplicación

Ejecutar pruebas

El **código de prueba** está separado del código del programa real y, en la mayoría de los IDE, los resultados / salida de la prueba también están separados de la salida del programa, lo que proporciona una estructura legible y conveniente.

Depuración y registros

La activación del registro del conector puede ayudarle a depurar muchos problemas con las aplicaciones del sistema de gestión de contenidos. Para ayudarle a aislar los problemas de una única aplicación, puede sustituir el nivel de registro actual al iniciar la aplicación.

Los programas de utilidad de registro de la API y el conector registran todas las excepciones, incluyendo las que no son errores. Ocasionalmente, pueden aparecer mensajes de error en el archivo de registro que no se propagan a los usuarios. En algunos casos, la API o la aplicación del usuario puede realizar la recuperación o continuar en el caso de avisos.

Java tiene dos gestores de registro: por omisión y log4j. Sólo puede configurar y utilizar uno de los gestores de registro cada vez. Se utiliza el mismo archivo de configuración, **cmblogconfig.properties**, para controlar el tipo de gestor de registro utilizado y la configuración específica para cada tipo de gestor de registro. Para obtener más información acerca del gestor de registro que desea utilizar, consulte la sección del archivo de configuración, **cmblogconfig.properties**, que pertenece al gestor de registro que desea utilizar.

Cuando se crea por primera vez la instancia del programa de utilidad de registro del conector, busca la vía de acceso de clases de la instancia de Java Virtual Machine para encontrar el archivo **ibmcmconfig.properties**. Después de encontrar este archivo, la API podrá buscar la ubicación de **cmblogconfig.properties**, el archivo de configuración de registro. Si no se localiza este archivo de configuración, se utilizan los valores de registro por omisión.

Errores de programación

Los **errores** en tiempo de ejecución son aquellos que ocurren de manera inesperada: disco duro lleno, error de red, división por cero, cast inválido, etc. Todos estos errores pueden ser manejados a través de excepciones. También hay errores debidos a tareas multihilo que ocurren en tiempo de ejecución y no todos se pueden controlar. Por ejemplo, un bloqueo entre hilos sería muy difícil de controlar y habría que añadir algún mecanismo que detecte esta situación y mate los hilos que corresponda.

Las excepciones son eventos que ocurren durante la ejecución de un programa y hacen que éste salga de su flujo normal de instrucciones. Este mecanismo permite tratar los errores de una forma elegante, ya que separa el código para el tratamiento de errores del código normal del programa. Se dice que una excepción es lanzada cuando se produce un error, y esta excepción puede ser capturada para tratar dicho error.

Tenemos diferentes tipos de excepciones dependiendo del tipo de error que representen. Todas ellas descienden de la clase Throwable, la cual tiene dos descendientes directos:

Error: Se refiere a errores graves en la máquina virtual de Java, como por ejemplo fallos al enlazar con alguna librería. Normalmente en los programas Java no se tratarán este tipo de errores.

Exception: Representa errores que no son críticos y por lo tanto pueden ser tratados y continuar la ejecución de la aplicación. La mayoría de los programas Java utilizan estas excepciones para el tratamiento de los errores que puedan ocurrir durante la ejecución del código.

Dentro de Exception, cabe destacar una subclase especial de excepciones denominada RuntimeException, de la cual derivarán todas aquellas excepciones referidas a los errores que comúnmente se pueden producir dentro de cualquier fragmento de código, como por ejemplo hacer una referencia a un puntero null, o acceder fuera de los límites de un array.

Estas RuntimeException se diferencian del resto de excepciones en que no son de tipo checked. Una excepción de tipo checked debe ser capturada o bien especificar que puede ser lanzada de forma obligatoria, y si no lo hacemos obtendremos un error de compilación. Dado que las **RuntimeException** pueden producirse en cualquier fragmento de código, sería impensable tener que añadir manejadores de excepciones y declarar que éstas pueden ser lanzadas en todo nuestro código. Deberemos:

Utilizar excepciones unchecked (no predecibles) para indicar errores graves en la lógica del programa, que normalmente no deberían ocurrir. Se utilizarán para comprobar la consistencia interna del programa.

Utilizar excepciones checked para mostrar errores que pueden ocurrir durante la ejecución de la aplicación, normalmente debidos a factores

externos como por ejemplo la lectura de un fichero con formato incorrecto, un fallo en la conexión, o la entrada de datos por parte del usuario.

Errores sintácticos

Un **error sintáctico** en el código de Java es una en la que el lenguaje que se utiliza para crear su código es incorrecto. Por ejemplo, si intenta crear una sentencia `if` que no incluye la condición entre paréntesis, incluso cuando la condición está presente en la misma línea que la sentencia `if`, eso es un error de sintaxis. El compilador detectará la mayoría de estos errores para usted. Si la sintaxis de su código es incorrecto, entonces en la mayoría de los casos, el compilador no puede utilizar el código para crear el código de bytes para el JRE.

- **El uso incorrecto de mayúsculas:** Uno de los errores más comunes de sintaxis que los nuevos desarrolladores hacen es capitalizar palabras clave, en lugar de usar minúsculas. Java es sensible a mayúsculas, así que usar el caso correcto al escribir su código es esencial. Este mismo error puede ocurrir con nombres de clases, variables, o cualquier otro código que escribe como parte de la aplicación Java. Una variable llamada `MiVar` es siempre diferente entre `myVar` nombrado.
- **División de una cadena de más de dos líneas:** En la mayoría de los casos, a Java no le importa si el código aparece en una o más líneas. Sin embargo, si se divide una cadena a través de líneas de modo que la cadena contiene un carácter de nueva línea, a continuación, el compilador se opondrá. La respuesta es poner fin a la cadena en la primera línea con una doble cita, agregue un signo más para indicar

al compilador que desea concatenar (añadir) esta cadena con otra cadena, y luego continuar la cadena en la siguiente línea.

- **Falta paréntesis:** Si usted hace una llamada a un método y no se incluye entre paréntesis después del nombre del método (incluso si usted no está enviando los argumentos del método), el compilador se registra un error.
- **Olvidarse de importar una clase:** Cada vez que desee utilizar una función de la API de Java en particular, debe importar la clase asociada a su aplicación. Por ejemplo, si su aplicación contiene Cadena de nombre de usuario, entonces debe agregar `java.lang.String`- importación para importar la clase `String`.
- **El tratamiento de un método estático como un método de instancia:** Los métodos estáticos son aquellos que están asociados con una clase específica, mientras que los métodos de instancia se asocian con un objeto creado a partir de la clase.
- **Missing llaves:** Cada vez que desee una característica de Java para aplicar a múltiples líneas de código, debe incluir toda la manzana entre llaves (`{}`). En la mayoría de los casos, el compilador detectará este error para usted. Por ejemplo, si usted trata de poner fin a una clase sin incluir la llave de cierre, el compilador generará un error. Este es un error que el compilador puede no mostrar la ubicación exacta del error porque no puede detectar donde falta el corchete -

simplemente sabe que falta uno. Este tipo de error también puede crear errores de ejecución. Por ejemplo, cuando se supone que una sentencia if para aplicar a múltiples líneas de código, pero dejar de lado las llaves, la sentencia if sólo afecta a la siguiente línea de código, y la aplicación funciona de forma incorrecta. Olvidar el nombre de clase o un objeto como parte de una llamada al método: Siempre se incluye la clase u objeto asociado a un método antes de hacer la llamada al método.

- **Olvidar el nombre de clase o un objeto como parte de una llamada al método:** Siempre se incluye la clase u objeto asociado a un método antes de hacer la llamada al método.
- **La omisión de una declaración de retorno:** Cuando se crea un método que se supone que devolver un valor y luego no proporcionan una sentencia return para devolver el valor, el compilador se quejará.
- **Mistyping la cabecera para el método main ():** El compilador no se quejará de este problema, pero verás inmediatamente cuando intenta iniciar la aplicación. Java se quejará de que no puede encontrar el método main ().

Errores de ejecución

Estos errores ocurren cuando la aplicación se está ejecutando, imagínate que cuando estás haciendo la presentación o probando tu aplicación en

público o frente a tu profesor y de repente se cuelga por lo general a este tipo de errores se los conoce como errores de compilación, ahora, por qué ocurren estos errores?, bueno, hay muchos factores desde los más básicos como por ejemplo:

- El usuario ingresa valores diferentes a los que la aplicación recibe
- Acceder a una posición en un arreglo la cual no existe.
- Almacenar cadenas donde se debe almacenar números
- Divisiones por cero.
- Digamos que en una aplicación móvil consuma datos de un servicio web y que al momento de consumir esos datos no haya conexión a internet, esto hace que la aplicación se cuelgue.

Invocaciones a funciones inexistentes

Una **invocación o llamada** a una función implica pasarle el control de la ejecución del programa, así como los argumentos o parámetros que requiere para realizar su tarea, se realiza colocando el nombre de la función y los argumentos actuales en el mismo orden que los parámetros formales correspondientes. La sintaxis del lenguaje permite también la invocación de funciones a través de punteros a funciones e incluso de referencias, aunque esto último sea menos frecuente. Cuando las funciones son miembros de clases la invocación sigue una sintaxis especial. En estos casos incluso existen operadores especiales para invocarlas a través de sus punteros.

Lectura o escritura de archivos

Java proporciona una cantidad de clases y métodos que le permiten leer y escribir archivos. Por supuesto, los tipos más comunes de archivos son archivos de disco. En Java, todos los archivos están orientados por bytes, y Java proporciona métodos para leer y escribir bytes desde y hacia un archivo.

Por lo tanto, leer y escribir archivos usando flujos de bytes es muy común. Sin embargo, Java le permite envolver un stream de archivos orientada a bytes dentro de un objeto basado en caracteres .

Para crear una stream de bytes vinculada a un archivo, se puede emplear **FileInputStream** o **FileOutputStream**. Para abrir un archivo, simplemente hay que crear un objeto de una de estas clases, especificando el nombre del archivo como un argumento para el constructor. Una vez que el archivo está abierto, se puede leer o escribir en él.

Interacción con MySQL u otras bases de datos

Cada propietario de base de datos implementa un driver JDBC que podemos utilizar en nuestras aplicaciones java. Normalmente habrá un driver por versión y tipo de base de datos, puesto que no siempre se cumple la compatibilidad hacia versiones anteriores.

El driver **JDBC** es el más básico de que dispone Java para acceder a la base de datos, por lo tanto utilizarlo implica:

- **Manejar manualmente las conexiones**, nos estamos arriesgando a no cerrar correctamente las conexiones a la base de datos, y un sin fin de problemas derivados.
- **Cada vez que escribamos una query SQL vamos a tener que escribirla completamente**, puesto que cualquier sentencia Insert, Update o Select requerirá escribir un gran número de campos.
- **Manejar manualmente las relaciones**, teniendo en cuenta el orden de inserción o modificación de tablas y columnas. Esto puede no parecer un gran problema, pero se complica exponencialmente en modelos de datos complejos.
- **El SQL lo escribiremos para la versión y tipo de nuestra base de datos**, por lo tanto, este código estará completamente acoplado, sin poder ser utilizado sin ser rehecho con otra base de datos.

Conexiones a redes de servicios

Cuando se hacen programas Java que se comuniquen lo habitual es que uno o varios actúen de cliente y uno o varios actúen de servidores.

- **Servidor:** espera peticiones, recibe datos de entrada y devuelve respuestas.
- **Cliente:** genera peticiones, las envía a un servidor y espera respuestas.

Un factor fundamental en los servidores es que tienen que ser capaces de procesar varias peticiones a la vez: deben ser **multihilo**.

Comprobar incorrectamente los datos de entradas

Puede prohibir o validar datos mediante reglas de validación al escribirlos en bases de datos de escritorio de Access. Puede usar el generador de expresiones para ayudarle a dar formato a la regla correctamente. Puede establecer reglas de validación en la vista de diseño de tabla o en la de hoja de datos de tabla. Hay tres tipos de reglas de validación en Access:

- **Regla de validación de campo:** Puede usar una regla de validación de campo para especificar un criterio que deben cumplir todos los valores de campo válidos. No debería tener que especificar el campo actual como parte de la regla a no ser que use el campo en una función. Las restricciones sobre los tipos de caracteres que se escriban en un campo pueden aplicarse más fácilmente con una máscara de entrada. Por ejemplo, un campo de fecha podría tener una regla de validación que no permita valores del pasado.
- **Regla de validación de registro:** Puede usar una regla de validación de registro para especificar una condición que necesitan cumplir todos los registros válidos. Puede comparar valores de distintos campos con una regla de validación de registro. Por ejemplo, un registro con dos campos de fecha puede necesitar que los valores de un campo siempre sean anteriores a los del otro
- **Validación en un formulario:** Puede usar la propiedad Regla de validación de un control de un formulario para especificar un criterio que deben cumplir todos los valores indicados para ese control. La

propiedad de control Regla de validación funciona como una regla de validación de campo. Normalmente, se usa una regla de validación de formulario en lugar de una regla de validación de campo si se trata de una regla específica del formulario y no de la tabla, sin importar donde se use.

Errores lógicos

Los errores lógicos en programación Java puede ser extremadamente difícil de encontrar, ya que no reflejan ningún tipo de problema de codificación o un error en el uso de los elementos del lenguaje Java. El código funciona perfectamente como está escrito - sólo no se está realizando la tarea que espera que éste realice. Como resultado, los errores lógicos pueden ser los errores más difíciles de encontrar. He aquí una lista de los errores lógicos comunes que enfrentan los desarrolladores de Java:

- **El uso de precedencia de operadores incorrecto:** El orden en que Java interpreta los operadores es importante. Aplicaciones a menudo producen un resultado erróneo porque el desarrollador no incluyó paréntesis en los lugares correctos.
- **Definición de la cuenta equivocada:** Posiblemente el error lógico más común está contando cosas incorrectamente. La gente está acostumbrada a las cuentas que comienzan con 1, y los ordenadores suelen comenzar el recuento con 0. Por lo tanto, no es raro encontrar que las aplicaciones son, precisamente, uno fuera en la realización de una tarea, si esa tarea se está ejecutando un bucle o trabajar con una colección de artículos.

- **Suponiendo una condición es verdadera cuando no lo es:** Los desarrolladores suelen leer la declaración se utiliza para definir una condición y asumir que la afirmación es verdadera (o falsa) sin verificar la lógica de la declaración.

El uso de una declaración o cuando realmente significaba utilizar una declaración también puede causar problemas. La lógica empleada para tomar decisiones hace que muchos desarrolladores, incluso los desarrolladores con experiencia, un montón de problemas. Siempre verifique sus supuestos de sentencias condicionales.

- **Basándose en los números de punto flotante para trabajos de precisión:** No se puede asumir que los números de punto flotante entregará un número específico. Esto significa que no se puede comprobar un número de punto flotante por la igualdad a cualquier valor específico - en su lugar debe utilizar un rango de valores para realizar la comprobación. Números de punto flotante son siempre una aproximación en Java.
- **Perder un punto y coma:** Es posible crear código Java que compila y funciona perfectamente bien a pesar de tener un punto y coma en el lugar equivocado.

Ayuda de depuración de variables

La depuración de un programa o debug permite ejecutar un programa de forma interactiva, paso a paso y examinar los valores de las variables. Esto proporciona valiosa información que permite comprobar el correcto funcionamiento de un programa o descubrir la causa de un error del que las trazas no proporcionan información suficiente. Los depuradores o debuggers son las aplicaciones que ejecutan el programa en modo depuración, permiten establecer puntos de ruptura, continuar la ejecución paso a paso o hasta el siguiente punto de ruptura e inspeccionar los valores de las variables.

Pocos programas están libres de errores, algunos errores se producen cuando se cumplen ciertas condiciones en los datos que maneja una aplicación. En ocasiones descubrir la causa de un error es suficiente si la aplicación emite trazas pero en los errores más complicados o con poca información se requiere depurar el programa.

La depuración se emplea en el momento de desarrollo o para reproducir y descubrir en el entorno de desarrollo local un error que se está produciendo en producción. Por la utilidad de la depuración todos los lenguajes ofrecen alguna posibilidad de hacer depuración. En Java la depuración se realiza configurando la máquina virtual con ciertos argumentos y el entorno integrado de desarrollo o IDE para conectarse a la máquina virtual.

El depurador permite añadir puntos de ruptura en cualquiera de las líneas del código fuente del programa para que la ejecución se detenga cuando se llegue a ese punto. Una vez detenida la ejecución el depurador permite examinar los valores de las variables disponibles en el ámbito de detención de la ejecución.

Para continuar la ejecución si se ha detenido en una línea de código con una llamada a una función el depurador ofrece la posibilidad de continuar entrando al código de esa función o saltar a la siguiente línea de código, también se ofrece la posibilidad de continuar la ejecución hasta la siguiente punto de ruptura. Los puntos de ruptura se pueden habilitar o deshabilitar, no siendo necesario eliminarlos para que no tengan efecto.

Niveles de informes de errores

Manejo de errores, mensajes de depuración, auditoría y archivos de log son diferentes aspectos del mismo tópico: como realizar un seguimiento de eventos dentro de una aplicación. A continuación, se expondrán las pautas para el manejo de excepciones. En términos generales, hay tres situaciones que hace que las excepciones sean lanzadas:

- **Excepciones que se producen en el código del cliente:** El código cliente intenta hacer algo que no está permitido por la API, de esta forma viola el contrato. El cliente puede tomar algún camino alternativo, si hay información útil proporcionada en la excepción. Por ejemplo: una excepción es lanzada cuando se está analizando un documento XML que no está bien formado. La excepción contiene información útil acerca de la localización en el documento XML donde se produce el problema. El cliente puede utilizar esta información para recuperarse del problema.
- **Excepciones por fallos en los recursos mantenidos:** Las excepciones se producen si existe un fallo derivado de un recurso. Por ejemplo, el agotamiento de memoria o el fallo de conexión cuando se cae una red. La respuesta del cliente a los recursos que fallan dependen del contexto. El cliente puede reintentar la

operación después de algún tiempo o simplemente registrar el fallo del recurso y detener la aplicación.

- **Excepciones por errores derivados del código programado:** En esta categoría, las excepciones se producen en la ejecución del código (como pueden ser `NullPointerException` e `IllegalArgumentException`). El código cliente usualmente no puede hacer nada con estos errores.

Java define dos tipos de excepciones:

- **Excepciones verificables:** Son aquellas que heredan de la clase `Exception`. El código cliente tiene que manejar estas excepciones lanzadas por la API mediante una cláusula `catch` o enviándola al método que llama mediante la cláusula `throws`.
- **Excepciones no verificables:** `RuntimeException` también es heredada de `Exception`. Sin embargo, todas las excepciones que heredan de `RuntimeException` tienen un tratamiento especial. No es necesario que el código cliente las maneje, y por esto se llaman no verificables.

Modificar los parámetros de informes de error

Los parámetros con valores de tabla se pueden utilizar en las modificaciones de datos basadas en conjuntos que afectan a varias filas mediante la ejecución de una única instrucción. Por ejemplo, puede seleccionar todas las filas de un parámetro con valores de tabla e insertarlas en una tabla de base de datos, o puede crear una instrucción

UPDATE combinando un parámetro con valores de tabla en la tabla que desea actualizar.

Existen varias limitaciones para los parámetros con valores de tabla:

No puede pasar parámetros con valores de tabla a funciones definidas por el usuario.

Los parámetros con valores de tabla solo se pueden indicar para admitir restricciones UNIQUE o PRIMARY KEY. SQL Server no mantiene estadísticas en los parámetros con valores de tabla.

Los parámetros con valores de tabla son de solo lectura en el código de Transact-SQL. No se pueden actualizar los valores de columna de las filas de un parámetro con valores de tabla y no se pueden insertar ni eliminar filas. Para modificar los datos que se pasan a un procedimiento almacenado o a una instrucción con parámetros en el parámetro con valores de tabla, debe insertar los datos en una tabla temporal o en una variable de tabla.

No se pueden usar instrucciones ALTER TABLE para modificar el diseño de parámetros con valores de tabla.

Puede transmitir objetos grandes en un parámetro con valores de tabla.

Desencadenar errores propios

Excepciones, o sencillamente problemas. En la programación siempre se producen errores, más o menos graves, pero que hay que gestionar y tratar correctamente. Por ello en java disponemos de un mecanismo consistente en el uso de bloques try/catch/finally. La técnica básica consiste en colocar las instrucciones que podrían provocar problemas dentro de un bloque try, y colocar a continuación uno o más bloques catch, de tal forma que si se

provoca un error de un determinado tipo, lo que haremos será saltar al bloque catch capaz de gestionar ese tipo de error específico. El bloque catch contendrá el código necesario para gestionar ese tipo específico de error. Suponiendo que no se hubiesen provocado errores en el bloque try, nunca se ejecutarían los bloques catch.

Solucionar errores

En programación esto sucede cuando podemos corregir errores de sintaxis y/o de lógica. Es decir, cuando sabemos depurar un programa. ¿Por qué? Pues simplemente porque implica que:

- Saber leer un programa
- Saber las reglas de sintaxis del lenguaje
- Entender la lógica de programación

- **Incompatyble types:** unexpected return value:

El mensaje de error indica que existen tipos incompatibles.

¿Qué hacer?

Revisar el tipo de método y la instrucción return si la incluye.

- **Invalid method declaration:** return type required

El mensaje de error indica que el método no ha sido declarado apropiadamente y que se requiere un tipo.

¿Qué hacer?

Revisar que el método tenga especificado un tipo de método, ya sea alguno de los tipos primitivos de datos o incluso en tipo void si no regresará valores.

- **Illegal start of expression**

Este mensaje de error aparece al inicio de un nuevo método, ya sea uno definido por el usuario o el mismo main. Y eso es debido a que un método anterior no se ha cerrado, cualquier inicio de otro método generará error de sintaxis.

¿Qué hacer?

Revisar que el método anterior tenga su llave de cierre.

- **Missing return statement**

Este error se genera simplemente porque se ha olvidado la sentencia return, la cual es obligatoria en los métodos de tipo int, char, float, double y boolean; además de cualquier tipo definido por el usuario.

¿Qué hacer?

Asegúrate que, si el método no es de tipo void, exista la sentencia de return.

- **class, interface or enum expected**

El mensaje explica que se espera el inicio de una clase, interface o enumeración en esa parte del código. ¿Por qué? Muy sencillo, porque la clase actual ya ha sido cerrada.

¿Qué hacer?

Revisar que en el método anterior no sobre una llave de cierre.

Registrar los errores en un archivo de registro

Los objetivos que se deben buscar al diseñar un mecanismo de control de excepciones deben ser:

- Evitar que la ocurrencia de excepciones se muestre al usuario final en pantalla de forma incontrolada.
- No perder información de las excepciones lanzadas a más bajo nivel en la aplicación.
- Mantener un Log de todas las excepciones que ocurren en el sistema y que sea lo más legible posible.
- Dar soporte al sistema de validaciones de la aplicación.

La estrategia planteada deberá ser en líneas generales la siguiente (planteamiento para JSF/ADF):

- Diseñar una jerarquía de excepciones especificando en qué casos se debe utilizar cada una de ellas.
- Establecer un conjunto de reglas para determinar que excepciones deben ser capturadas. Unas recomendaciones podrían ser:
 - Las excepciones controladas deben ser capturadas en el método en el que se producen, transformadas en las

excepciones de aplicación adecuadas y enviadas hacia las capas superiores.

- Las excepciones de tipo no controladas que se lancen en la capa de acceso a datos deberán ser capturadas en la capa de servicio y traducidas en una excepción de tipo Aplicación.
- Las excepciones de tipo no controlado que se produzcan en la capa de servicio se considerarán en general errores del sistema. Para evitar que su ocurrencia se muestre al usuario se tendrá que utilizar el mecanismo de control de errores Web.
- En la capa de vista / controlador se tendrán que capturar tanto las excepciones de tipo Aplicación que provienen de las capas inferiores como las que se produzcan en esta capa. Las excepciones serán capturadas en el backing bean dentro de cada uno de los métodos donde se ejecuta el acceso a la capa de servicio.
- El sistema deberá gestionar los mensajes orientados a los usuarios y a los administradores. Los primeros serán los que se muestren en las páginas de la aplicación y los segundos los que se registren en el Log del sistema.
- Si se considera necesario, se tendrá que adaptar el mecanismo de ejecución de las acciones de los backing beans, para que todas compartan el mismo comportamiento respecto a las excepciones.

19. DISEÑO FINAL DE LA WEB

Diseño de logo final

Para el **logo de nuestra web** finalmente se ha optado por una imagen minimalista, añadiendo además el nombre de nuestra aplicación:



Diseño general de la Web

Para el **diseño general de la web** hicimos un primer boceto, sobre él fuimos añadiendo los diferentes elementos y funcionalidades que expondremos a continuación.

Barra Superior

En la **barra superior** se han incluido las siguientes funcionalidades.

- **Fecha, hora y año.**
- **Botón de Registro.** Para realizar la creación de la cuenta de usuario.
- **Botón de Inicio.** Para realizar el inicio de sesión una vez la cuenta de usuario se haya generado.
- **Botón de Ayuda.** Nos brinda información general sobre las posibles incidencias en la página web.

Botón de Registro

Al clicar el **botón de registro** nos envía a una ventana en la que podemos indicar todos los datos del usuario a registrar.

A screenshot of a web registration form titled "Registro:" in a black header bar. The form is centered on a page with a grey left sidebar and a yellow right sidebar. The form itself has a white background with a yellow dashed border. It contains six input fields: "NAME", "E-MAIL", "telefono", "password", "poner otra vez el password", and a "GO!" button at the bottom.

Se completan los siguientes campos:

- Nombre del usuario
- E-mail del usuario
- Teléfono del usuario
- Contraseña seleccionada
- Repetición de contraseña seleccionada

Botón de inicio de Sesión

Al clicar el **botón de inicio de sesión** nos redirige a la ventana de apuestas.

FÚTBOL									
RESULTADO		1	x	2					
21:00	Elche CF Rayo Vallecano	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	+5	+10	+50	v	<input type="text"/>
16:00	FC Cartagena Real Valladolid	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	+5	+10	+50	v	<input type="text"/>
20:45	Juventus Torino	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	+5	+10	+50	v	<input type="text"/>
19:00	Sevilla Madrid	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	+5	+10	+50	v	<input type="text"/>
	RF Espana	—	—	—					

Desde este apartado podremos ver la siguiente información:

- **Hora de los partidos que tendrán lugar.**
- **Partidos enfrentados.**
- **Apuestas a realizar.** Local (1), Empate (x), Visitante (2).
- **Importe de la apuesta.** Aquí podremos indicar si queremos hacer un importe cerrado de (5, 10 o 50€) o si queremos indicar un importe diferente que indicaremos en la casilla derecha y verificaremos con el check que aparece en la izquierda de la casilla.

Botón de Ayuda

Al clicar el **botón de ayuda** nos dirige hacia una ventana en la que se intentan solventar algunas de las incidencias más frecuentes en la web.

FAQ:

¿Cómo hacer apuestas deportivas?

Es muy sencillo: entra en la página de apuestas Barbas, regístrate y deposita aprovechando alguna de las múltiples ofertas de las páginas de apuestas y ya podrás apostar.

¿Cómo funcionan las apuestas deportivas?

Cada participante en un evento deportivo tiene una cuota que se fija en función a la probabilidad estimada de que ello ocurra. Es decir, la cuota y por lo tanto el beneficio que se obtiene con una victoria del Real Madrid será menor si se enfrenta a un equipo de Segunda división que si juega la final de la Champions, ya que es más probable que ocurra.

¿Qué es el hándicap en las apuestas?

¿Qué es una apuesta múltiple?

Es lo mismo que una apuesta combinada, la cuota de cada pronóstico se multiplica por las de las otras selecciones.

¿Qué son las apuestas cruzadas?

Estas son apuestas en las que los apostantes definen la apuesta, y la casa de apuestas deportivas simplemente actúa como intermediario, garantizando el pago y cobrando una pequeña comisión por ello.

¿Qué es el yield en las apuestas?


Es el rendimiento general o beneficio medio con relación a la inversión, que tiene un apostador sobre el total de sus apuestas.

¿Qué son las unidades en las apuestas?

Es la cantidad que decidimos invertir en una apuesta, que suele equivaler a un 1% del dinero disponible para ello. No hay que confundirlo con el stake, que siempre se mantiene entre el 1/10 y el 10/10, las unidades aumentan a medida que aumenta el bankroll.


¿Cómo hacer apuestas seguras?

Las apuestas deportivas seguras están basadas en el principio financiero de arbitraje, que consiste en coger el valor de ambas variables (en este caso cuotas) por encima de 2, de tal forma que gane quien gane, se obtiene beneficio.




Bonos y Promociones

Elegibilidad para las promociones
Cómo solicitar bonos individuales
Ver todos




Depositos

Métodos de Depósito
Verificación de identidad
El «Cajero»
Ver todos



Juegos

Reglas de apuestas deportivas
Cómo usar una apuesta gratuita
Guano/resultados del juego
Ver todos



Conectar

Cambio de contraseña
¿Has olvidado tu contraseña?
¿Has olvidado tu usuario?
Ver todos

Zona Central de la Web

En la **parte central de la web** hemos incluido información que orienta al usuario sobre los diferentes métodos de apuestas para que le sea más sencillo apostar con una guía sencilla.



APUESTAS BÁRBARAS

! Disfruta de la oferta de apuestas deportivas !

En **APUESTAS BÁRBARAS** encontrarás una extensa oferta de apuestas futbolísticas online. Elige de entre las diferentes Ligas y marca un buen gol con tus apuestas de fútbol.

Elige cualquier competición de fútbol y haz tus apuestas en cualquiera de las **Ligas Europeas**, entre semana y el fin de semana, haz tus apuestas de segunda división si es que buscas buenas cuotas, porque es una categoría muy igualada en la que ningún favorito es claro y los resultados de partidos no están claros. Si te gustan las apuestas a largo plazo, busca en los deportes las apuestas a ganador de una competición o de un galardón en concreto.

Como ya sabrás, en España cada casa de apuestas online necesita una licencia para operar y Apuestas Bárbaras, por supuesto, es una **Casa de apuestas con Licencia en España**. Haz todas las apuestas que quieras en un entorno seguro y regulado, en el que no tendrás problemas de ningún tipo para depositar o retirar tu dinero. A Apuestas Bárbaras le

pero podría ocurrir que se tuviera que doblar la apuesta demasiadas veces seguidas.

El Método 2/3

Consiste en escoger **3 partidos** y **hacer 3 apuestas combinando solo 2 de las selecciones**, que deben tener una cuota cercana a 2.0 para que sea rentable.

Apostamos 1? a victorias de Equipo X y Equipo Y (cuota 1.75)

Apostamos 1? a victorias de Equipo X y Equipo Z (cuota 1.75)

Apostamos 1? a victorias de Equipo Y y Equipo Z (cuota 1.75)

Inversión: 3? ? Premio acertado 2 de 3: 3.5? ? Beneficio: 0.5?

Sistema Patent

Es un sistema parecido bastante desconocido, pero que si se realiza correctamente ofrece bastantes

Zona Inferior de la Web

En la parte inferior se ha añadido publicidad de nuestra web.



Dinero fácil y RÁPIDO

La única página de apuestas que te ayuda a ganar dinero.

///////// Regístrate y obtén Barbara Coin

Además se han incorporado tres botones que incluyen información sobre los **métodos de pago**.



Métodos de Pago Aceptados

A continuación encontrarás una lista y una breve descripción de cada método de depósito ofrecido por **Apuestas Bárbaras**. Para consultar las instrucciones completas sobre cómo realizar un depósito con alguno de los métodos enumerados en la lista.

Métodos de depósito con los que también puedes realizar retiros	
Tarjetas de crédito y débito VISA	Un método fácil y rápido para depositar directamente en tu cuenta. Disponible para socios de todo el mundo.
MasterCard	Un método fácil y rápido para depositar directamente en tu cuenta.
ApplePay	Una tecnología de pago y una de las funciones de los últimos iPhones, iPads y Apple Watch. Recoge información de tus tarjetas de crédito, tarjetas de débito y otros datos de

