Turismo Argentina

Índice:

Descripción: Turismo Argentina es una aplicación web realizada en Java la cual simula un e-commerce de turismo con roles de administrador y visitante.

Patrón de diseño: Esta aplicación fue realizada siguiendo el patrón de diseño MVC (Model View Controller, o Modelo Vista Controlador en español).

Base de datos utilizada: MySQL.

Recursos externos utilizados:

- GliderJS:
    1. glider.min.js
    2. glider.min.css

- Fontawesome.

- Fancybox.
    1. fancybox.umd.js
    2. fancybox.css

Recursos propios utilizados:

- myscript.js:
    1. Es el archivo de JavaScript que alberga el algoritmo y funciones del menú de navegación responsive, y el algoritmo/configuración necesaria para el carrusel realizado con GliderJS.
    2. Ubicación: …\\PaginaTurismo\\src\\main\\webapp\\script\\myscript.js

- Imágenes:
    1. Imágenes de personas (1.png, 2.png, 3.png y 4.png).
    2. cielo.jpg
    3. fondo.jpg
    4. Ubicación: …\\PaginaTurismo\\src\\main\\webapp\\img

- Estilos en cascada (documentos CSS):
    1. estilos.css: Hoja de estilos en cascada principal de la aplicación.
    2. detalles.css: Es la hoja de estilos en cascada de la página de mostrarLugar.jsp y de todos los JSP que esta incluye (los que se encuentran en la carpeta detalles). (…\\PaginaTurismo\\src\\main\\webapp\\WEB-INF\\paginas\\detalles\\mostrarLugar).
    3. actividades.css: Es la hoja de estilos en cascada de la página de mostrarActividad.jsp y de todos los JSP que esta incluye (los que se encuentran en la carpeta actividades). (…\\PaginaTurismo\\src\\main\\webapp\\WEB-INF\\paginas\\actividades\\mostrarActividad).
    4. sesion.css: Es la hoja de estilos en cascada todos los JSP ubicados en la carpeta sesión, en los que destacan panel.jsp (el panel del administrador), sesion.jsp (página para iniciar sesión) y registrarse.jsp (página para registrarse en la aplicación). (…\\PaginaTurismo\\src\\main\\webapp\\WEB-INF\\paginas\\sesion).

- Otros recursos:
    1. favicon.ico: Es el ícono de la aplicación ubicado al lado del nombre de la página en la que se encuentra el usuario.
    2. logoAzul.svg y logoBlanco.svg: Son vectores que se utilizan como logos/imágenes en diferentes páginas de la aplicación web.

Dependencias:

- mysql-connector.
- Librería de apache commons.

- iTextPDF.

Paquetes Java:

- Paquete de dominio: Contiene las clases de entidad, las cuales representan un registro en la base de datos.
    1. Lugar.java
    2. Actividad.java
    3. Usuario.java
    4. Cargo.java
    5. Contacto.java

- Paquete de datos: Contiene las clases e interfaces DAO, las cuales se encargan de realizar las operaciones con la base de datos. A su vez posee la clase Conexión.
    1. Conexion.java
    2. ILugarDao.java (interface) y LugarDao.java (implementación).
    3. IActividadDao.java (interface) y ActividadDao.java (implementación).
    4. IUsuarioDao.java (interface) y UsuarioDao.java (implementación).
    5. IContactoDao.java (interface) y ContactoDao.java (implementación).

- Paquete web: Contiene los Servlets que harán de controlador en el modelo MVC. Se encargan de la lógica del programa para así hacer que los JSP solo desplieguen información.
    1. ServletControlador.java: Es el servlet principal de la aplicación web.
    2. ServletImagen.java: Se encarga de controlar el procesamiento de las imágenes para listarlas en los JSP correspondientes.
    3. ServletPanel.java: Es el servlet encargado de controlar todas las acciones lógicas de la página panel.jsp.

Lugar.java

- Atributos:

```
private int idLugar;
private String nombre;
private String descripcion;
private InputStream portada;
private InputStream foto1;
private InputStream foto2;
private InputStream foto3;
private double precio;
```

- Constructores:

```
public Lugar() {

}
```

```
public Lugar(int idLugar) {
    this.idLugar=idLugar;
}
```

```
public Lugar(String nombre, String descripcion, InputStream portada, InputStream foto1, InputStream foto2, InputStream foto3, double precio) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.portada = portada;
        this.foto1 = foto1;
        this.foto2 = foto2;
        this.foto3 = foto3;
        this.precio = precio;
    }
```

```
public Lugar(int idLugar, String nombre, String descripcion, InputStream portada, InputStream foto1, InputStream foto2, InputStream foto3, double precio) {
        this.idLugar = idLugar;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.portada = portada;
        this.foto1 = foto1;
        this.foto2 = foto2;
        this.foto3 = foto3;
        this.precio = precio;
    }
```

- Métodos getter y setter para todos sus atributos.
- Método toString (para revisar si se necesita el estado de objeto lugar).

Actividad.java

- Atributos:

```
private int idActividad;
private String nombre;
private InputStream imagen;
private double precio;
```

- Constructores:

```
public Actividad() {

}
```

```
public Actividad(int idActividad) {
    this.idActividad=idActividad;
}
```

```
public Actividad(String nombre, InputStream imagen, double precio) {
    this.nombre = nombre;
    this.imagen = imagen;
    this.precio = precio;
}
```

```
public Actividad(int idActividad, String nombre, InputStream imagen, double precio) {
    this.idActividad = idActividad;
    this.nombre = nombre;
    this.imagen = imagen;
    this.precio = precio;
}
```

- Métodos getter y setter para cada uno de sus atributos.
- Método toString para revisar el estado de un objeto Actividad en todo momento.

Usuario.java

- Atributos:

```
private int idUsuario;
private String nombre;
private String email;
private String contraseña;
private Cargo cargo;
```

- Constructores:

```
public Usuario() {

}
```

```
public Usuario(String nombre, String email, String contraseña) {
    this.nombre=nombre;
    this.email=email;
    this.contraseña=contraseña;
}
```

```
public Usuario(int idUsuario, String nombre, String email, String contraseña, Cargo cargo) {
    this.idUsuario = idUsuario;
    this.nombre = nombre;
    this.email = email;
    this.contraseña = contraseña;
    this.cargo = cargo;
}
```

- Métodos getter y setter para cada uno de sus atributos.
- Método toString para revisar el estado de un objeto Usuario en el momento que se desee.

Cargo.java

- Atributos:

```
private int idCargo;
private String nombre;
```

- Constructores:

```
public Cargo() {

}
```

```
public Cargo(String nombre) {
    this.nombre=nombre;
}
```

- Métodos getter y setter para cada uno de sus atributos.

Contacto.java

- Atributos:

```
private int idContacto;
    private String nombre;
    private String email;
    private String comentario;
```

- Constructores:

```
public Contacto() {

    }
```

```
public Contacto(int idContacto) {
        this.idContacto=idContacto;
    }
```

```
public Contacto(String nombre, String email, String comentario) {
        this.nombre=nombre;
        this.email=email;
        this.comentario=comentario;
    }
```

```
public Contacto(int idContacto, String nombre, String email, String comentario) {
        this.idContacto=idContacto;
        this.nombre=nombre;
        this.email=email;
        this.comentario=comentario;
    }
```

- Métodos getter y setter para cada uno de sus atributos.
- Método toString para revisar el estado de un objeto Contacto cuando se desee.

Conexión.java

- Atributos:

```
private static final String JDBC_URL =
"jdbc:mysql://localhost:3306/pagina_turismo?useSSL=false&useTimezone=true&s
erverTimezone=UTC&allowPublicKeyRetrieval=true";

private static final String JDBC_USER = "root";

private static final String JDBC_PASSWORD = "";

private static BasicDataSource dataSource;
```

- Creación del pool de conexiones:

Retorna un objeto DataSource, no recibe argumentos.

```
public static DataSource getDataSource() {
      if (dataSource == null) {
         dataSource = new BasicDataSource();
         dataSource.setUrl(JDBC_URL);
         dataSource.setUsername(JDBC_USER);
         dataSource.setPassword(JDBC_PASSWORD);
         dataSource.setInitialSize(50);
      }
      return dataSource;
   }
```

Primero pregunta si el atributo dataSource esta instanciado, si no, lo instancia y le coloca la URL, el Username, el Password y si valor inicial con los datos de los otros 3 atributos y un valor (50) que representa la cantidad de conexiones (es variable). Finalmente devuelve el objeto dataSource.

- Método getConnection:

Retorna un objeto Connection, no recibe argumentos.

Lo único que hace este método es retornar el objeto Connection obtenido por el método getConnection utilizando el método getDataSource.

```
public static Connection getConnection() throws SQLException {
    return getDataSource().getConnection();
}
```

- Método close:

Este método no retorna nada, sin embargo, esta sobrescrito, por ende, recibe hasta 3 argumentos diferentes en cada implementación.

Lo único que hace es, con un bloque try/catch, cierra el objeto del argumento que se le pasa. Ejemplo:

```
public static void close(ResultSet rs) {
    try {
       rs.close();
    } catch (SQLException ex) {
       ex.printStackTrace(System.out);
    }
}
```

El argumento puede ser del tipo ResultSet (como en el ejemplo), PreparedStatement o Connection.

LugarDao.java

- Atributos (consultas a la base de datos):

```
private static final String SQL_SELECT = "SELECT idlugar, nombre, descripcion, portada, foto1, foto2, foto3, precio FROM lugar";

private static final String SQL_SELECT_BY_ID = "SELECT idlugar, nombre, descripcion, portada, foto1, foto2, foto3, precio FROM lugar WHERE idlugar = ?";

private static final String SQL_INSERT = "INSERT INTO lugar(nombre, descripcion, portada, foto1, foto2, foto3, precio) VALUES(?, ?, ?, ?, ?, ?, ?)";

private static final String SQL_UPDATE = "UPDATE lugar SET nombre=?, descripcion=?, portada=?, foto1=?, foto2=?, foto3=?, precio=? WHERE idlugar=?";

private static final String SQL_DELETE = "DELETE FROM lugar WHERE idlugar=?";
```

- Método listar:

Retorna una lista de objetos Usuario. No recibe argumentos.

Primeramente, instancia un objeto Lugar llamado lugar, y un ArrayList del tipo List de objetos Lugar, a su vez, prepara la conexión.

Mediante un bloque try/catch, ejecuta la consulta SQL_SELECT y con un bucle while recupera todos los parámetros obtenidos en la consulta, los coloca como valores del objeto lugar creado anteriormente, y lo añade a la lista de lugares. Esto se repite (en el bucle while) mientras que la consulta SQL haya seguido dando resultados.

Finalmente, cierra las conexiones y retorna la lista lugares.

- Método encontrar:

Retorna un objeto Lugar y recibe de argumento un objeto Lugar llamado lugar.

Primeramente, prepara las conexiones y con un bloque try/catch ejecuta la consulta SQL_SELECT_BY_ID, pasándole como único parámetro el id obtenido del objeto Lugar.

Recupera todos los valores obtenidos en la consulta y se los coloca al objeto lugar. Finalmente, cierra las conexiones y retorna el objeto lugar.

- Método insertar:

No retorna nada, pero recibe de argumento un objeto Lugar.

Prepara la conexión y, con un bloque try/catch, ejecuta la consulta SQL_INSERT, enviándole como parámetros los valores de los atributos del objeto Lugar (nombre, descripción, portada, etc.). Cabe recalcar que este método ejecuta el método executeUpdate() del objeto PreparedStatement en lugar de executeQuery() como los anteriores. Finalmente cierra las conexiones.

- Método actualizar:

No retorna nada, pero recibe de argumento un objeto Lugar.

Prepara la conexión y, con un bloque try/catch, ejecuta la consulta SQL_UPDATE, pasándole como parámetros los valores de los atributos del objeto Lugar. Al igual que el anterior, este método ejecuta el método executeUpdate() en vez de executeQuery(). Finalmente cierra las conexiones.

- Método eliminar:

No retorna nada, pero recibe de argumento un objeto Lugar.

Prepara las conexiones y, con un bloque try/catch, ejecuta la consulta SQL_DELETE, pasándole como único parámetro el valor del atributo id del objeto lugar. Utiliza executeUpdate() (igual que los dos anteriores). Finalmente, cierra las conexiones.

- Método escribirImagen:

No retorna nada. Recibe de argumentos una variable del tipo int llamada id, un objeto HttpServletResponse llamado resp, y una variable del tipo int llamada num.

Prepara las conexiones y los objetos para realizar una consulta, a su vez, crea un objeto InputStream, un objeto OutputStream, un objeto BufferedInputStream y un objeto BufferedOutputStream, los cuales iguala a null.

A su vez, utilizando el objeto resp, llama al método setContentType, pasándole como argumento un String que tiene como valor “image/*”.

Con un bloque try/catch, iguala el objeto OutputStream al valor obtenido con el método getOutputStream del objeto resp, y ejecuta la consulta SQL_SELECT_BY_ID, pasándole como parámetro el int id que venía como argumento en el método.

Si la consulta retorna un registro, utiliza un switch utilizando la variable num como llave.

Si el valor de la llave es 1, el objeto InputStream es igual al valor obtenido por el parámetro “portada” de la consulta, si el valor es 2, es igual al parámetro “foto1”, si el valor es 3, es igual al parámetro “foto2”, y si el valor es 4, es igual al parámetro “foto3”.

Posteriormente, inicializa el objeto BufferedInputStream pasándole como argumento el objeto InputStream e inicializa el objeto BufferedOutputStream pasándole como argumento el objeto OutputStream.

Crea una variable int llamada i y, mediante un bucle while, ejecuta el método write del objeto BufferedOutputStream pasándole como argumento la variable i. El bucle while se repite mientras que i sea diferente de -1, siendo que iguala i al valor obtenido por el método read del objeto BufferedInputStream.

Finalmente cierra todas las conexiones y objetos.

ActividadDao.java

- Atributos (consultas a la base de datos):

```
private static final String SQL_SELECT = "SELECT idactividad, nombre, imagen, precio FROM actividad";

private static final String SQL_SELECT_BY_ID = "SELECT idactividad, nombre, imagen, precio FROM actividad WHERE idactividad = ?";

private static final String SQL_SELECT_BY_RANDOM_ID = "SELECT idactividad, nombre, imagen, precio FROM actividad WHERE idactividad IN(?,?,?)";

private static final String SQL_INSERT = "INSERT INTO actividad(nombre, imagen, precio) VALUES(?, ?, ?)";

private static final String SQL_UPDATE = "UPDATE actividad SET nombre=?, imagen=?, precio=? WHERE idactividad=?";

private static final String SQL_DELETE = "DELETE FROM actividad WHERE idactividad=?";
```

- Método listar:

Retorna un ArrayList del tipo List de objetos Actividad. No recibe argumentos.

Prepara la conexión y los objetos necesarios para ejecutar la consulta a la base de datos, además de declarar un objeto Actividad y una lista de objetos Actividad.

Con un bloque try/catch ejecuta la consulta SQL_SELECT y con un bucle while recupera todos los valores de los parámetros de la consulta y se los asigna al objeto Actividad, posteriormente añade este objeto a la lista.

Finalmente, cierra las conexiones y retorna la lista de objetos Actividad.

- Método generar:

Retorna un ArrayList del tipo List de objetos Actividad. No recibe argumentos.

Define 3 variables de tipo int con el nombre de n1, n2 y n3 respectivamente, con un ciclo do while, les asigna un valor aleatorio a las 3 (siendo únicamente posible un valor

entre 1 y el tamaño de la lista obtenida al utilizar el método listar de la misma clase) hasta que ninguna coincida.

Prepara la conexión y declara un objeto Actividad y una lista de objetos Actividad.

Posteriormente prepara la conexión y, con un bloque try/catch, ejecuta la consulta SELECT_BY_RANDOM_ID, pasándole como parámetros las 3 variables de tipo int.

En un bloque try/catch, ejecuta la consulta y en un bucle while, el cual se ejecuta mientras que la consulta haya regresado registros, recupera los valores de los parámetros de la consulta y se los asigna al objeto Actividad, el cual añade a la lista de objetos Actividad.

Finalmente cierra la conexión y retorna la lista de objetos Actividad.

- Método encontrar:

Retorna un objeto Actividad y recibe como argumento un objeto de este mismo tipo.

Prepara la conexión y, mediante un bloque try/catch, ejecuta la consulta SQL_SELECT_BY_ID, pasándole como parámetro el valor del id del objeto Actividad.

Posteriormente, ejecuta la consulta y recupera todos los valores de los parámetros de esta, asignándoselos al objeto Actividad.

Finalmente, cierra las conexiones y retorna el objeto Actividad.

- Método insertar:

No retorna nada, pero recibe como argumento un objeto Actividad.

Prepara la conexión y, mediante un bloque try/catch, ejecuta la consulta SQL_INSERT, pasándole los valores del objeto Actividad como parámetro de la consulta. Ejecuta la actualización con el método executeUpdate del objeto PreparedStatement y cierra todas las conexiones.

- Método actualizar:

No retorna nada, pero recibe como argumento un objeto Actividad.

Prepara la conexión y, mediante un bloque try/catch ejecuta la consulta SQL_UPDATE, pasándole como parámetro los valores de los atributos del objeto Actividad. Ejecuta la actualización con el método executeUpdate y cierra todas las conexiones.

- Método eliminar:

No retorna nada, pero recibe como argumento un objeto Actividad.

Prepara la conexión y, mediante un bloque try/catch ejecuta la consulta SQL_DELETE, pasándole como único valor el atributo id del objeto Actividad. Ejecuta la actualización y cierra las conexiones.

- Método escribirImagen:

No retorna nada. Recibe como argumentos una variable de tipo int llamada id y un objeto HttpServletResponse.

Prepara la conexión a la base de datos y los objetos requeridos para hacer la consulta a su vez, crea un objeto InputStream, un objeto OutputStream, un objeto BufferedInputStream y un objeto BufferedOutputStream, los cuales iguala a null.

A su vez, utilizando el objeto resp, llama al método setContentType, pasándole como argumento un String que tiene como valor “image/*”.

Si la consulta retorna un registro, iguala el objeto InputStream al valor obtenido del parámetro “imagen” de la consulta.

Posteriormente, inicializa el objeto BufferedInputStream pasándole como argumento el objeto InputStream e inicializa el objeto BufferedOutputStream pasándole como argumento el objeto OutputStream.

Crea una variable int llamada i y, mediante un bucle while, ejecuta el método write del objeto BufferedOutputStream pasándole como argumento la variable i. El bucle while se repite mientras que i sea diferente de -1, siendo que iguala i al valor obtenido por el método read del objeto BufferedInputStream.

Finalmente cierra todas las conexiones y objetos.

UsuarioDao.java

- Atributos (consultas a la base de datos):

```
private static final String SQL_SELECT = "SELECT idusuario, nombre, email, contraseña, cod_cargo FROM usuario";

private static final String SQL_INSERT = "INSERT INTO usuario(nombre, email, contraseña, cod_cargo) VALUES(?, ?, ?, ?)";

private static final String SQL_SELECT_BY_NAME = "SELECT idusuario, nombre, email, contraseña, cod_cargo FROM usuario WHERE nombre=?";

private static final String SQL_SELECT_BY_EMAIL = "SELECT idusuario, nombre, email, contraseña, cod_cargo FROM usuario WHERE email=?";
```

-Método identificar:

Retorna un objeto usuario, recibe como argumento un objeto Usuario llamado usuario.

Este método posee un String llamado SQL que contiene lo siguiente:

```
String SQL="SELECT u.idusuario, c.nombre FROM usuario u INNER JOIN cargo c ON u.idusuario=c.idcargo WHERE u.nombre='" + usuario.getNombre() AND u.contraseña='"+ usuario.getContraseña() + "'";
```

Posteriormente, el método prepara la conexión a la base de datos y crea un objeto Usuario con el nombre usuarioVerificado. Con un bloque try catch ejecuta la consulta SQL (la variable String SQL) y si devuelve un valor, instancia el objeto usuarioNuevo, coloca su id al valor de “idusuario” de la consulta, coloca su nombre al nombre del atributo usuario, crea un nuevo objeto Cargo para usuarioNuevo y se lo coloca con el valor “nombre” obtenido en la consulta. Si no devuelve un valor, retorna nulo.

Finalmente, cierra las conexiones y retorna el objeto usuarioVerificado.

- Método insertar:

Retorna un entero (el cual representa el número de registros afectados en la base de datos) y recibe como argumento un objeto Usuario llamado usuario.

El método prepara la conexión a la base de datos y crea una variable de tipo int llamada cantidad, igualándola a 0.

Posteriormente, con un bloque try/catch, ejecuta la consulta preparada SQL_INSERT pasándole los valores de nombre, email y contraseña del atributo usuario, y como cuarto parámetro le pasa el valor de 2, el cual representa un Cargo para el objeto Usuario (en la base de datos es el cargo de visitante). Iguala la variable cant a lo que devuelve ejecutar la consulta, cierra las conexiones y retorna esta variable.

- Método listar:

Retorna un ArrayList de Usuario. No recibe argumentos.

Este método prepara la conexión a la base de datos e instancia un objeto Usuario llamado usuario y un ArrayList de Usuario llamado usuarios, luego, con un bloque try/catch, ejecuta la consulta SQL_SELECT y con un bucle while, recupera todos los valores de los registros (nombres, contraseñas, emails, etc.) y se las coloca al objeto usuario previamente creado. Si el código de cargo obtenido en la consulta es 1, el cargo de ese usuario se coloca como administrador, si es 2, como visitante.

Por último, añade la variable usuario a la lista usuarios, se repite el ciclo hasta que ya no haya más registros encontrados por la consulta, cierra las conexiones y retorna el ArrayList usuarios.

- Método verificarNombre:

Retorna un boolean, recibe de argumento un objeto Usuario llamado usuario.

Primeramente, prepara la conexión a la base de datos y establece una variable boolean llamada bandera.

Con un bloque try/catch, realiza la consulta SQL_SELECT_BY_NAME configurándole su único parámetro con nombre del usuario. Si la consulta devuelve un resultado, se coloca el valor de la variable bandera como true.

Finalmente, cierra las conexiones y retorna la variable bandera.

- Método verificarEmail:

Realiza los mismos procedimientos que el método verificarNombre, pero utilizando la consulta SQL_SELECT_BY_EMAIL y pasándole como parámetro el valor de email del objeto usuario.

ContactoDao.java

- Atributos (consultas a la base de datos):

```
private static final String SQL_SELECT = "SELECT idcontacto, nombre, email, comentario FROM contacto";

private static final String SQL_SELECT_BY_ID = "SELECT idcontacto, nombre, email, comentario FROM contacto WHERE idcontacto=?";

private static final String SQL_INSERT = "INSERT INTO contacto(nombre, email, comentario) VALUES(?, ?, ?)";

private static final String SQL_DELETE = "DELETE FROM contacto WHERE idcontacto=?";
```

- Método listar:

Retorna una lista de objetos Contacto. No recibe argumentos.

Prepara la conexión y los objetos necesarios para hacer una consulta a la base de datos, además de declarar un objeto Consulta y una lista de objetos Consulta.

Mediante un bloque try/catch, ejecuta la consulta SQL_SELECT y con un bucle while, el cual se ejecuta mientras la consulta haya devuelto valores, recupera todos valores de los parámetros de la consulta y se los asigna al objeto Contacto, el cual añade a la lista de objetos Contacto posteriormente.

Finalmente, cierra las conexiones y retorna la lista de objetos Contacto.

- Método encontrar:

Retorna un objeto Contacto. Recibe como argumento un objeto Contacto.

Prepara la conexión y, mediante un bloque try/catch, ejecuta la consulta SQL_SELECT_BY_ID, pasándole como parámetro el valor del atributo id del objeto Contacto que recibe como argumento.

Posteriormente, recupera todos los valores de los parámetros de la consulta y se los asigna al objeto Contacto.

Finalmente, cierra las conexiones y retorna el objeto Contacto.

- Método insertar:

Retorna una variable de tipo int (la cual significa la cantidad de registros modificados) y recibe como argumento un objeto de tipo Contacto.

Prepara la conexión e inicializa una variable de tipo int llamada cant.

Mediante un bloque try/catch, ejecuta la consulta SQL_INSERT pasándole como valores los atributos del objeto Contacto, los cuales son nombre, email y comentario. La variable cant la iguala al resultado de ejecutar el método executeUpdate del objeto PreparedStatement.

Finalmente, cierra las conexiones y retorna la variable cant.

- Método eliminar:

No retorna nada. Recibe un objeto Contacto como argumento.

Prepara la conexión y el objeto PreparedStatement y, mediante un bloque try/catch, ejecuta la consulta SQL_DELETE, pasándole como parámetro el valor del atributo id dedl objeto Contacto.

Ejecuta la consulta mediante el método executeUpdate del objeto PreparedStatement y cierra las conexiones.

ServletControlador.java

- Atributos:

```
private final ILugarDao datosL;
private final IActividadDao datosA;
private final IUsuarioDao datosU;
private final IContactoDao datosC;
```

- Constructor:

```
public ServletPanel() {
    this.datosL = new LugarDao();
    this.datosA = new ActividadDao();
    this.datosU = new UsuarioDao();
    this.datosC = new ContactoDao();
}
```

- Método doGet(): Primero, recupera el parámetro acción del alcance de request y se lo asigna a una variable de tipo String llamada acción, posteriormente, si la variable acción no es nula, con un Switch analiza todos los posibles casos del valor de la variable acción. Si la variable acción es nula, llama al método accionDefault(req, resp) ubicado en el mismo Servlet.
  Casos del Switch:
  1. Caso “mostrar”: Llama al método mostrarLugar(req, resp) ubicado en el mismo Servlet.
  2. Caso “listar”: Llama al método mostrarActividades(req, resp) ubicado en el mismo Servlet.
  3. Caso “sesion”: Llama al método mostrarInicioSesion(req, resp) ubicado en el mismo Servlet.
  4. Caso “registrarse”: Llama al método mostrarRegistro(req, resp) ubicado en el mismo Servlet.

5. Caso “cerrar”: Llama al método cerrarSesion(req, resp) ubicado en el mismo Servlet.
6. Caso “formularioLugar”: Llama al método mostrarFormularioLugar(req, resp) ubicado en el mismo Servlet.
7. Caso “formularioActividad”: Llama al método mostrarFormularioActividad (req, resp) ubicado en el mismo Servlet.
8. Caso “carrito”: Llama al método mostrarCarrito(req, resp) ubicado en el mismo Servlet.
9. Caso “quitarLugarCarrito”: Llama al método quitarLugarCarrito(req, resp) ubicado en el mismo Servlet.
10. Caso “quitarActividadCarrito”: Llama al método quitarActividadCarrito(req, resp) ubicado en el mismo Servlet.
11. Caso “descargarFactura”: Llama al método descargarFactura(req, resp) ubicado en el mismo Servlet.
12. Caso por default: Llama al método accionDefault(req, resp) ubicado en el mismo Servlet.

- Método doPost(): Primero, recupera el parámetro acción del alcance de request y se lo asigna a una variable de tipo String llamada acción, posteriormente, si la variable acción no es nula, con un Switch analiza todos los posibles casos del valor de la variable acción. Si la variable acción es nula, llama al método accionDefault(req, resp) ubicado en el mismo Servlet.
  Casos del Switch:
  1. Caso “verificar”: Llama al método verificar(req, resp) ubicado en el mismo Servlet.
  2. Caso “registrarse”: Llama al método registrarse(req, resp) ubicado en el mismo Servlet.
  3. Caso “enviarContacto”: Llama al método insertarContacto(req, resp) ubicado en el mismo Servlet.
  4. Caso “agregarLugarCarrito”: Llama al método agregarLugarCarrito(req, resp) ubicado en el mismo Servlet.
  5. Caso “agregarActividadCarrito”: Llama al método agregarActividadCarrito (req, resp) ubicado en el mismo Servlet.
  6. Caso por default: Llama al método accionDefault(req, resp) ubicado en el mismo Servlet.

- Métodos:
  1. accionDefault(req, resp):

Este método establece dos ArrayList, uno del objeto Lugar, con el nombre lugares, y otro del objeto Actividad, llamado actividades. Estos ArrayList los instancia con el método listar del atributo datosL y el método generar del atributo datosA respectivamente.
Luego obtiene/crea (si no está creada) la sesión (llamada sesion) y le coloca como atributo ambos ArrayList con los nombres de "lugares" y "actividades" respectivamente. A su vez, le coloca como atributo llamado "mensajeContacto" el valor obtenido al utilizar el método getAttribute con el valor "mensajeContacto" utilizando el objeto de request.
Por último, utiliza el objeto resp (response) para redireccionar la página a principal.jsp.

2. calcularTotal: Devuelve una variable de tipo double, recibe como parámetros una lista de objetos Lugar y una de objetos Actividad.
   Establece una variable del tipo double llamada total, con valor 0.
   Posteriormente, recorre la lista de objetos Lugar, sumándole a la variable total el precio de cada objeto Lugar en cada iteración.
   Luego hace lo mismo con la lista de objetos Actividad y retorna la variable total.

3. actualizarPanel: Recupera la sesión y crea 4 ArrayList del tipo List, los cuales son:
   1- List<Usuario> usuarios, al cual iguala al método listar del atributo datosU.
   2- List<Lugar>lugares, al cual iguala al método listar del atributo datosL.
   3- List<Actividad>actividades, la cual iguala al método listar del atributo datosA.
   4- List<Contacto>contactos, la cual iguala al método listar del atributo datosC.
   Posteriormente, le agrega a la sesión los siguientes 6 atributos:
   1- Key: "lugares" Valor: lugares.
   2- Key: "actividades" Valor: actividades.
   3- Key: "contactos" Valor: contactos.
   4- Key: "totalUsuarios" Valor: usuarios.size().
   5- Key: "totalLugares" Valor: lugares.size().
   6- Key: "totalActividades" Valor: actividades.size().
   Finalmente, redirecciona la página a WEB-INF/paginas/sesion/panel.jsp.

- Métodos doGet:

1. mostrarLugar: Recupera el parámetro “idLugar” con el objeto Request y luego crea un objeto Lugar pasándole este mismo valor, utilizando el constructor que únicamente pide el id.
   Luego le asigna al objeto Lugar el valor de lo obtenido al utilizar el método encontrar del atributo datosL, pasándole como parámetro el mismo objeto Lugar.
   Posteriormente, le asigna al objeto Request el atributo de “lugar” con el valor del objeto Lugar, y redirecciona la página a /WEB-INF/paginas/detalles/ mostrarLugar.jsp.

2. mostrarActividades: Crea un ArrayList del tipo List de objetos Actividad y lo iguala al valor obtenido del método listar, del atributo datosA.
   Posteriormente, le asigna el atributo “actividades” al objeto Request con el valor de la lista de objetos Actividad, y redirecciona la página a /WEB-INF/paginas/actividades/ mostrarActividades.jsp.

3. mostrarInicioSesion:
   Redirecciona la página a WEB-INF/paginas/sesion/sesion.jsp.

4. mostrarRegistro:
   Redirecciona la página a WEB-INF/paginas/sesion/registrarse.jsp.

5. cerrarSesion: Obtiene el objeto HttpSession mediante el método getSession del objeto Request, le coloca el atributo de “usuario” con valor null, la invalida con el método invalidate, y, mediante el objeto Response, redirecciona al index.jsp.

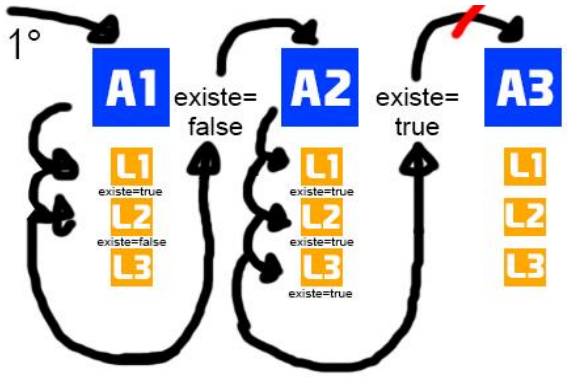
6. mostrarFormularioLugar: Recupera el parámetro “idLugar” con el objeto Request y crea un objeto Lugar pasándole como parámetro del constructor este id.
   Posteriormente, utiliza el método encontrar del atributo datosL pasándole como parámetro el objeto Lugar. Obtiene la sesión y le asigna como atributo este objeto Lugar. Por último, redirecciona a WEB-INF/paginas/formulariosCarrito/formularioLugar.jsp

7. mostrarFormularioActividad: Recupera el parámetro “idActividad” con el objeto Request y crea un objeto Actividad pasándole como parámetro del constructor este id.
   Posteriormente, utiliza el método encontrar del atributo datosA pasándole como parámetro el objeto Actividad. Obtiene la sesión y le asigna como

atributo este objeto Actividad. Por último, redirecciona a WEB-INF/paginas/formulariosCarrito/formularioActividad.jsp.

8. mostrarCarrito: Recupera el objeto HttpSession y crea dos ArrayList del tipo List, uno de objetos Lugar, y otro de objetos Actividad, los cuales iguala a los atributos “lugaresCarrito” y “actividadesCarrito” de la sesión respectivamente.
   Si la lista de objetos Lugar es nula, la instancia y la coloca como atributo a la sesión, lo mismo con la lista de objetos Actividad.
   Luego, aplica un algoritmo para verificar si alguna actividad turística no posee su sitio turístico definido en el carrito. El algoritmo funciona tal que así:
   Primero, crea dos variables, una del tipo boolean llamada existe, y otra del tipo String llamada mensaje, la cual iguala a “” (vacío).
   Posteriormente, si ambas listas de objetos no están vacías, recorre la lista de objetos Actividad y, por cada objeto Actividad, recorre la lista de objetos Lugar.
   En cada iteración de la lista de objetos Lugar pregunta si el nombre del objeto Actividad NO posee el nombre del objeto Lugar. En este caso, coloca el valor de la variable existe en true.
   En caso contrario, coloca la variable existe en false y rompe el ciclo.
   Posterior al recorrido de la lista de objetos Lugar, mientras itera el bucle de objetos Actividades, pregunta si existe es igual a true, y si lo es, rompe el ciclo.
   Posterior a ambos ciclos de repetición, si la variable existe tiene como valor true, le coloca como valor a la variable mensaje un aviso de advertencia y la coloca como atributo en el objeto Request.
   Finalizado el algoritmo, el método crea una variable del tipo double llamada total, la cual iguala al método calcularTotal del Servlet. Coloca esta variable como atributo del objeto Request y redirecciona la página a WEB-INF/paginas/index/carrito.jsp.

   Ejemplo del algoritmo (suponiendo que haya 3 actividades en la lista de objetos Actividad y 3 lugares en la lista de objetos Lugar):



9. quitarLugarCarrito: Recupera la sesión y el parámetro “idLugar” del objeto Request.
   Posteriormente, crea un objeto Lugar pasándole este parámetro.
   Luego crea una lista de objetos Lugar con el valor del atributo “lugaresCarrito” de la sesión y, con el método removeIf, elimina de la lista el objeto Lugar que tenga el mismo id que el objeto Lugar creado con el parámetro “idLugar”.
   Por último, le coloca como atributo a la sesión la lista de objetos Lugar y ejecuta el método mostrarCarrito.

10. quitarActividadCarrito: Realiza los mismos procedimientos que el método quitarLugarCarrito, pero con un objeto Actividad y una lista de objetos Actividad.

11. descargarFactura: Este método utiliza la librería de iTextPDF.
    Primero recupera la lista de objetos Lugar y la lista de objetos Actividad, las cuales son atributos de la sesión. Después, verifica que alguna no este vacía. Sí es así, escribe el PDF con la información correspondiente y lo muestra en pantalla, si no, establece una variable String que contiene un mensaje de advertencia, ya que el carrito este vacío, y lo coloca como atributo del objeto Request. Posteriormente ejecuta el método mostrarCarrito.

- Métodos doPost:
  1. verificar: Primero, crea un objeto Usuario y recupera los valores del formulario sesion.jsp (WEB-INF/paginas/sesion/sesion.jsp), asignándoselos al objeto Usuario (nombre y contraseña).
     Después, iguala el objeto Usuario al resultado del método identificar del atributo datosU, al cual se le pasa como parámetro el mismo objeto Usuario.
     Posteriormente, se ejecuta un condicional con 3 condiciones:
     1- Si el objeto Usuario no es nulo y el nombre del cargo del usuario es igual a “ADMINISTRADOR”: Recupera la sesión y le coloca como atributo de

nombre “administrador” el objeto Usuario. Por último, ejecuta el método actualizarPanel, ubicado en el mismo Servlet.
2- Si el objeto Usuario no es nulo y el nombre del cargo del usuario es igual a “VISITANTE”: Recupera la sesión y le coloca como atributo de nombre “visitante” el objeto Usuario. Por último, ejecuta el método accionDefault, ubicado en el mismo Servlet.
3- Si ninguna de las anteriores 2 condiciones se cumple, crea una variable String llamada mensaje cuyo valor es un código de error y se la asigna como atributo al objeto Request con el nombre de “mensaje”. Por último, redirecciona la página a WEB-INF/paginas/sesion/sesion.jsp.

2. Método registrarse: Recupera la sesión y también los parámetros “password” y “confpassword” del formulario de registrarse.jsp (WEB-INF/paginas/sesion/registrarse.jsp).
   Si estos dos últimos parámetros tienen la misma cadena de valor, también recupera el parámetro de “usuario” y “email” del mismo formulario.
   Posteriormente, crea un objeto Usuario pasándole como argumento los parámetros de nombre, email y password.

   Después chequea que el método verificarNombre o el método verificarEmail del atributo datosU no regresen true, pasándoles a ambos como argumento el objeto Usuario. Si alguno de los dos métodos regresa true, entonces establece una variable String llamada mensaje cuyo valor es una cadena de error debido a que el nombre de usuario o el mail ya están registrados en la aplicación. Este mensaje lo establece como atributo del objeto Request y recarga la página.
   Si ninguno de los dos métodos regresa true, recupera la sesión y utiliza el método insertar del atributo datosU pasándole como parámetro el objeto Usuario. Por último, agrega como atributo de la sesión la key “visitante” con el valor del objeto Usuario y ejecuta el método accionDefault.

   Si las cadenas de los parámetros “password” y “confpassword” no coinciden, declara una variable String llamada mensaje la cual contiene una cadena de error y la establece como atributo del objeto Request, recargando la página.

3. Método insertarContacto: Recupera la sesión y se fija si el atributo “mensajeContacto” de la sesión es nulo.
   Si lo es, recupera los parámetros “nombre”, “email” y “comentario” del formulario seccionContacto (WEB-INF/paginas/index/seccionContacto.jsp). Con estos parámetros crea un objeto Contacto y utiliza el método insertar del atributo datosC pasándole este mismo objeto.

Por último, establece una variable mensaje del tipo String a la cual le pasa el valor de una cadena de confirmación de envío y lo agrega como atributo del objeto Request. Posteriormente ejecuta el método accionDefault.
Si el atributo "mensajeContacto" no es nulo, establece una variable de tipo String con el valor de una cadena explicándole al usuario que ya mando un mensaje hace poco tiempo. Por último, agrega como atributo al objeto Request este mensaje y ejecuta el método accionDefault.

4. Método agregarLugarCarrito: Recupera la sesión y, a su vez, recupera los parámetros de "idLugar" y de "cantidad" del objeto Request.
   Inicializa un objeto Lugar pasandole como parámetro el idLugar recuperado. Iguala este objeto Lugar a lo retornado por el método encontrar del atributo datosL pasándole como parámetro el objeto Lugar.
   Posteriormente, configura el atributo precio del objeto Lugar multiplicando su valor por el parámetro cantidad.
   Después establece una lista de objetos Lugar cuyo valor es el atributo "lugaresCarrito" de la sesión.
   Si la lista es nula, la inicializa como un nuevo ArrayList.
   Establece 2 variables, una de tipo String llamada mensaje, cuyo valor es "" (vacío), y otra de tipo boolean llamada existe.
   Recorre la lista de objetos Lugar y, si el objeto Lugar tiene el mismo nombre que alguno de los objetos Lugar que ya se encuentra en la lista, le agrega a la variable existe el valor de true.
   Por último, si la variable existe no es verdadera, añade el objeto Lugar previamente creado a la lista de objetos Lugar. Si la variable no es verdadera, se le agrega de valor a la variable mensaje una cadena especificándole al cliente que el lugar seleccionado ya se encuentra en el carrito.
   Se le agrega el atributo de "mensaje" con el valor de la variable mensaje al objeto Request y el atributo "lugaresCarrito" con el valor de la lista de objetos Lugar a la sesión y, posteriormente, se ejecuta el método mostrarCarrito.

5. Método agregarActividadCarrito: Realiza los mismos procedimientos que el método anterior, reemplazando los objetos Lugar por objetos Actividad y utilizando los métodos pertinentes.

ServletImagen.java

- Atributos:

```
private final ILugarDao datosL;
private final IActividadDao datosA;
```

- Constructor:

```
public ServletImagen() {
    this.datosL = new LugarDao();
    this.datosA = new ActividadDao();
}
```

Este Servlet solo maneja método doGet, no doPost u otros.

- Método doGet: Primero, recupera el parámetro acción del alcance de request y se lo asigna a una variable de tipo String llamada acción, posteriormente, si la variable acción no es nula, con un Switch analiza todos los posibles casos del valor de la variable acción. A su vez define dos variables de tipo int llamadas id y num. La variable id tiene de valor el parámetro "id" del objeto Request.
  Casos del Switch:
  1. Caso "listarPortada": Iguala la variable num a 1 y llama al método escribirImagen del atributo datosL, pasándole los valores de id, el objeto Response y num.
  2. Caso "listarFoto1": Realiza los mismos procedimientos que el caso "listarPortada", pero igualando la variable num a 2.
  3. Caso "listarFoto2": Realiza los mismos procedimientos que los anteriores 2 casos, pero igualando la variable num a 3.
  4. Caso "listarFoto3": Realiza los mismos procedimientos que los anteriores 3 casos, pero igualando la variable num a 4.
  5. Caso "listarActividad": Ejecuta el método escribirImagen del atributo datosA, pasándole los valores de id y el objeto Response.

ServletPanel.java

- Atributos:

```
private final ILugarDao datosL;
private final IActividadDao datosA;
private final IUsuarioDao datosU;
private final IContactoDao datosC;
```

- Constructor:

```
public ServletPanel() {
    this.datosL = new LugarDao();
    this.datosA = new ActividadDao();
    this.datosU = new UsuarioDao();
    this.datosC = new ContactoDao();
}
```

- Método doGet: Primero, recupera el parámetro acción del alcance de request y se lo asigna a una variable de tipo String llamada acción, posteriormente, si la variable acción no es nula, con un Switch analiza todos los posibles casos del valor de la variable acción. Si la variable acción es nula, llama al método accionDefault(req, resp) ubicado en el mismo Servlet.
  Casos del Switch:
  1. Caso “editarLugar”: Llama al método editarLugar(req, resp) ubicado en el mismo Servlet.
  2. Caso “agregarLugar”: Llama al método agregarLugar(req, resp) ubicado en el mismo Servlet.
  3. Caso “eliminarLugar”: Llama al método eliminarLugar(req, resp) ubicado en el mismo Servlet.

4. Caso “mostrarActividad”: Llama al método mostrarActividad(req, resp) ubicado en el mismo Servlet.
5. Caso “editarActividad”: Llama al método editarActividad(req, resp) ubicado en el mismo Servlet.
6. Caso “agregarActividad”: Llama al método agregarActividad(req, resp) ubicado en el mismo Servlet.
7. Caso “eliminarActividad”: Llama al método eliminarActividad(req, resp) ubicado en el mismo Servlet.
8. Caso “mostrarContacto”: Llama al método mostrarContacto(req, resp) ubicado en el mismo Servlet.
9. Caso “eliminarContacto”: Llama al método eliminarContacto(req, resp) ubicado en el mismo Servlet.
10. Caso por default: Llama al método accionDefault(req, resp) ubicado en el mismo Servlet.

- Métodos doPost: Primero, recupera el parámetro acción del alcance de request y se lo asigna a una variable de tipo String llamada acción, posteriormente, si la variable acción no es nula, con un Switch analiza todos los posibles casos del valor de la variable acción. Si la variable acción es nula, llama al método accionDefault(req, resp) ubicado en el mismo Servlet.
  Casos del Switch:
  1. Caso “modificarLugar”: Llama al método actualizarLugar(req, resp) ubicado en el mismo Servlet.
  2. Caso “insertarLugar”: Llama al método insertarLugar(req, resp) ubicado en el mismo Servlet.
  3. Caso “modificarActividad”: Llama al método actualizarActividad(req, resp) ubicado en el mismo Servlet.
  4. Caso “insertarActividad”: Llama al método insertarActividad(req, resp) ubicado en el mismo Servlet.
  5. Caso por default: Llama al método accionDefault(req, resp) ubicado en el mismo Servlet.

- Métodos:
  1. accionDefault: Recupera la sesión y crea 4 ArrayList del tipo List, los cuales son:
     1- List<Usuario> usuarios, al cual iguala al método listar del atributo datosU.
     2- List<Lugar>lugares, al cual iguala al método listar del atributo datosL.
     3- List<Actividad>actividades, la cual iguala al método listar del atributo datosA.

4- List<Contacto>contactos, la cual iguala al método listar del atributo datosC.

Posteriormente, le agrega a la sesión los siguientes 6 atributos:

1- Key: “lugares” Valor: lugares.

2- Key: “actividades” Valor: actividades.

3- Key: “contactos” Valor: contactos.

4- Key: “totalUsuarios” Valor: usuarios.size().

5- Key: “totalLugares” Valor: lugares.size().

6- Key: “totalActividades” Valor: actividades.size().

Finalmente, redirecciona la página a WEB-INF/paginas/sesion/panel.jsp.

- Métodos doGet:
    1. editarLugar: Recupera el parámetro “idLugar” del objeto Request y crea un objeto Lugar cuyo valor es el retorno del método encontrar del atributo datosL pasándole como parámetro un nuevo objeto Lugar con el idLugar como parámetro.
       Posteriormente le asigna al objeto Request el atributo de “lugar” con el valor del objeto Lugar y redirecciona la página a WEB-INF/paginas/sesion/lugar/editarLugar.jsp.

    2. agregarLugar: Redirecciona la página a WEB-INF/paginas/sesion/lugar/agregarLugar.jsp.

    3. eliminarLugar: Recupera el parámetro “idLugar” del objeto Request y crea un objeto Lugar con este parámetro. Posteriormente, ejecuta el método eliminar del atributo datosL pasándole como parámetro el objeto Lugar. Por último, ejecuta el método accionDefault del mismo Servlet.

    4. mostrarActividad: Recupera el parámetro “idActividad” del objeto Request y crea un objeto Actividad cuyo valor es el retorno del método encontrar del atributo datosA pasándole como parámetro un nuevo objeto Actividad con el idActividad como parámetro.
       Posteriormente le asigna al objeto Request el atributo de “actividad” con el valor del objeto Actividad y redirecciona la página a /WEB-INF/paginas/sesion/actividad/mostrarActividad.jsp.

    5. editarActividad: Realiza los mismos pasos que el método anterior, pero redirecciona la página a WEB-INF/paginas/sesion/actividad/editarActividad.jsp.

6. agregarActividad: Redirecciona la página a WEB-INF/paginas/sesion/actividad/agregarActividad.jsp.

7. eliminarActividad: Recupera el parámetro “idActividad” del objeto Request y crea un objeto Actividad con este parámetro. Posteriormente, ejecuta el método eliminar del atributo datosA pasándole como parámetro el objeto Actividad. Por último, ejecuta el método accionDefault del mismo Servlet.

8. mostrarContacto: Recupera el parámetro “idContacto” del objeto Request y crea un objeto Contacto cuyo valor es el retorno del método encontrar del atributo datosC pasándole como parámetro un nuevo objeto Contacto con el idContacto como parámetro.
   Posteriormente le asigna al objeto Request el atributo de “contacto” con el valor del objeto Contacto y redirecciona la página a WEB-INF/paginas/sesion/contacto/mostrarContacto.jsp.

9. eliminarContacto: Recupera el parámetro “idContacto” del objeto Request y crea un objeto Contacto con este parámetro. Posteriormente, ejecuta el método eliminar del atributo datosC pasándole como parámetro el objeto Contacto. Por último, ejecuta el método accionDefault del mismo Servlet.

- Métodos doPost:
  1. actualizarLugar: Recupera todos los parámetros del formulario editarLugar.jsp (WEB-INF/paginas/sesion/lugar/editarLugar.jsp) y se los asigna a un nuevo objeto Lugar. Posteriormente, ejecuta el método actualizar del atributo datosL pasándole como parámetro el objeto Lugar. Finalmente ejecuta el método accionDefault del mismo Servlet.

  2. insertarLugar: Recupera todos los parámetros del formulario agregarLugar.jsp (WEB-INF/paginas/sesion/lugar/agregarLugar.jsp) y se los asigna a un nuevo objeto Lugar. Posteriormente, ejecuta el método insertar del atributo datosL pasándole como parámetro el objeto Actividad. Finalmente ejecuta el método accionDefault del mismo Servlet.

  3. actualizarActividad: Recupera todos los parámetros del formulario editarActividad.jsp (WEB-INF/paginas/sesion/actividad/editarActividad.jsp) y se los asigna a un nuevo objeto Actividad. Posteriormente, ejecuta el método actualizar del atributo datosA pasándole como parámetro el objeto Actividad. Finalmente ejecuta el método accionDefault del mismo Servlet.

4. insertarActividad: Recupera todos los parámetros del formulario agregarActividad.jsp (WEB-INF/paginas/sesion/actividad/agregarActividad.jsp) y se los asigna a un nuevo objeto Actividad. Posteriormente, ejecuta el método insertar del atributo datosA pasándole como parámetro el objeto Actividad. Finalmente ejecuta el método accionDefault del mismo Servlet.