PROYECTO FINAL.
Programando un proceso de negocio y acceso a base de datos con JDBC

Objetivo:
El objetivo de este proyecto es adquirir los conocimientos básicos para implementar un proceso de negocio de un caso de estudio empleando los conceptos de desarrollo orientado a interfaces y desarrollo de aplicaciones multicapa haciendo uso de los conceptos que se han revisado a lo largo del semestre.

1. Procesos de negocio.

A nivel general un proceso de negocio lo podemos definir como un conjunto de tareas relacionadas y ordenadas que se deben ejecutar para lograr un objetivo establecido. Dentro del desarrollo de sistemas de software, estos procesos de negocio se definen durante la etapa de requerimientos. Por ejemplo, para el caso de estudio de CENAT, podemos definir varios procesos de negocio:

- Proceso: Preparar una nueva obra de teatro para ponerla en escena.
- Proceso: Programar la exhibición de una obra de teatro a nivel nacional.
- Proceso: Calcular las ganancias de una obra de teatro a nivel nacional durante una determinada temporada.
- Proceso: Administrar el catálogo de centros de exhibición y sus correspondientes salas.
- Proceso: Registro de actores (registro de sus datos, su foto, y los datos dependiendo de su tipo).

Cada uno de estos procesos a su vez, se integran por actividades más pequeñas que en su conjunto permiten ejecutar el proceso de negocio de forma exitosa. En este documento se presenta el desarrollo para implementar el proceso de negocio "*Preparar una nueva obra de teatro para ponerla en escena*".

2. Implementación de un proceso de negocio.

2.1. *Identificando actividades*

El primer punto a identificar, son las actividades que integran al proceso de negocio. Es decir, para poder preparar una obra de teatro para ser exhibida, se deberán completar las siguientes acciones:

I. Registrar la nueva obra de teatro en el catálogo con estatus REGISTRADA con su correspondiente histórico.
II. Registrar los personajes de la obra de teatro.
III. Asignar a cada actor el personaje que representará, o en su caso registrar a un nuevo actor en caso de que se requiera.
IV. Una vez que se han registrado todos los actores y personajes, actualizar el estatus de la obra a "EN PREPARACION", con su correspondiente estatus.
V. En este periodo de tiempo, los actores ensayan, y se pueden generar cambios de actor.
VI. Finalmente al terminar los ensayos, se cambia el estatus de la obra a LISTA PARA PUBLICAR, con su correspondiente histórico.

Para poder realizar estas tareas, es necesario que el sistema proporcione la interfaz necesaria para interactuar con la base de datos. En el caso de un sistema Web, a través de paginas WEB, o en su defecto a través de alguna aplicación de escritorio.

2.2. *Diseño de la aplicación*

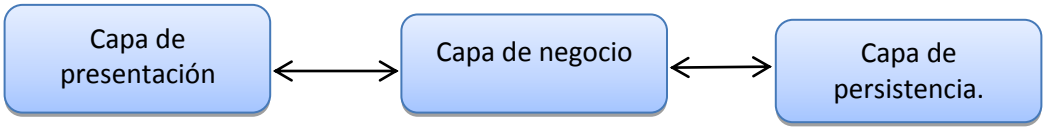
El siguiente paso es realizar el diseño de la aplicación ¿Qué componentes se deben desarrollar?, ¿Cómo se comunicarán las paginas web con la base de datos?, ¿Cuál será la forma mas adecuada de implementar este proceso de negocio?

Una de las principales técnicas que se emplean, en especial en sistemas web es el uso del concepto de sistemas multicapa empleando un diseño orientado a interfaces, el cual se describe a continuación.

Sistemas multicapa.

A nivel básico, típicamente, un sistema multicapa esta formado por 3 capas:

- **Capa de presentación.** En ella se encuentran todos los componentesgráficos, pantallas, controladores, paginas web (en el caso de un sistema web), etc. que permiten la interacción del usuario final con el sistema.
- **Capa de negocio.** En ella se encuentran las clases que se encargan de ejecutar y validar toda la lógica que un proceso de negocio requiere. En esta capa se realizan cálculos, se procesan los datos que son enviados por la capa de presentación.
- **Capa de persistencia.** Es la encargada de interactuar con la base de datos, realiza las operaciones que le indica la capa de negocio. (inserciones, actualizaciones, consultas, etc.)



Como se puede observar, existe una comunicación entre capas para poder ejecutar un proceso de negocio. Las siguientes reglas se deben respetar en un diseño de este estilo:

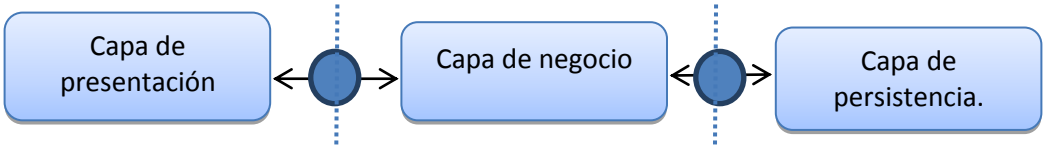
- La comunicación solo es entre capas adyacentes, es decir, no sería valido que la capa de presentación se comunique con la capa de persistencia.

- El intercambio de información entre capa y capa se realiza a través de clases, típicamente llamadas clases de negocio o entidades. Por ejemplo, la clase `ObraTeatro`, `Actor`, `Personaje`, `Sala`. Todas estas clases en su conjunto representan el modelo orientado a objetos de las entidades de una aplicación, en la mayoría de los casos, dichas entidades tienen su correspondencia con la base de datos. (Clase `Actor`, tabla `ACTOR`). Por ejemplo, si se desea registrar a un nuevo actor, la capa de presentación le pasará una instancia de la clase `Actor` a la capa de negocio con los datos capturados en una página web. En la capa de negocio, se realizan validaciones, se procesa la información, por ejemplo, validar requisitos del actor para poder ser registrado, etc., y finalmente, si todo es correcto, envía el objeto validado a la capa de persistencia para que este sea almacenado en la base de datos. Las entidades de negocio típicamente son clases que contienen todos los atributos de una entidad y sus métodos de acceso (clase encapsulada).

- Para implementar cada capa, existe una gran variedad de tecnologías. Por ejemplo, para implementar la capa de presentación se puede emplear Java Swing , o si es Web, JSPs, o frameworks mas desarrollados como Struts, SpringMVC, Rubi, Grails, etc. Para la capa de negocios, por ejemplo, puede ser desarrollada empleando EJBs, Spring, etc. Y para la parte de persistencia Hibernate, JDBC, JPA, etc. El punto importante es que la comunicación entre capas no debe depender de una tecnología en particular. Por ejemplo, si se cambia la estrategia de persistencia de JDBC a Hibernate, las demás capas no deben modificarse en lo absoluto. Para lograr este desacoplamiento entre capas se emplea el concepto de diseño orientado a interfaces.

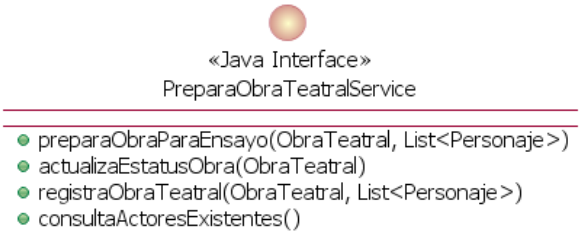
Diseño orientado a interfaces.

A nivel básico, este concepto permite desacoplar capas adyacentes permitiendo la comunicación entre ellas sin que existan dependencias particulares de la forma en la que una capa se implementa. Para lograr esto, se agregan interfaces entre la capa de presentación y la de negocio, y otra mas entre la capa de negocio y la capa de persistencia.



- A las interfaces que se definen entre la capa de presentación y la de negocio, se le conocen como Servicios, o interfaces de Servicios.
- A las interfaces que se definen entre la capa de negocio y la de persistencia se le conoce como DAO (Data Access Object), tomando la idea del patrón DAO para acceso a datos.

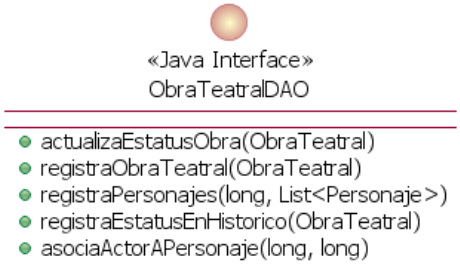
Recordando el concepto de interface, esta solo puede contener métodos abstractos. En este contexto, los métodos representan las operaciones que pueden ser invocadas por una capa adyacente. Por ejemplo, para el proceso de negocio en cuestión, la interface o "Servicio" que atenderá las peticiones de la capa de presentación se muestran en el siguiente diagrama.



La secuencia de eventos es la siguiente:

I. El usuario captura los datos de una nueva obra de teatro junto con sus personajes y se invoca al método `registraObraTeatral` para validar e invocar su persistencia en la base de datos.
II. Se solicita una consulta de los actores existentes para poder seleccionar los actores que representarán a cada personaje de la obra empleando el método `consultaActoresExistentes`.
III. Una vez que el usuario selecciona a los actores, se invoca al método `preparaObraParaEnsayo`, el cual es el encargado de invocar la persistencia de la asociación realizada Personaje – Actor. Para finalmente invocar el cambio de status de la Obra a `EN PREPARACION`.

La siguiente interface muestra la definición de un DAO, que es invocado por este servicio de negocio. Observar que los métodos realizan pequeñas operaciones en la base de datos. Cada método debe hacer solo lo que le corresponde, insertar, modificar, etc.



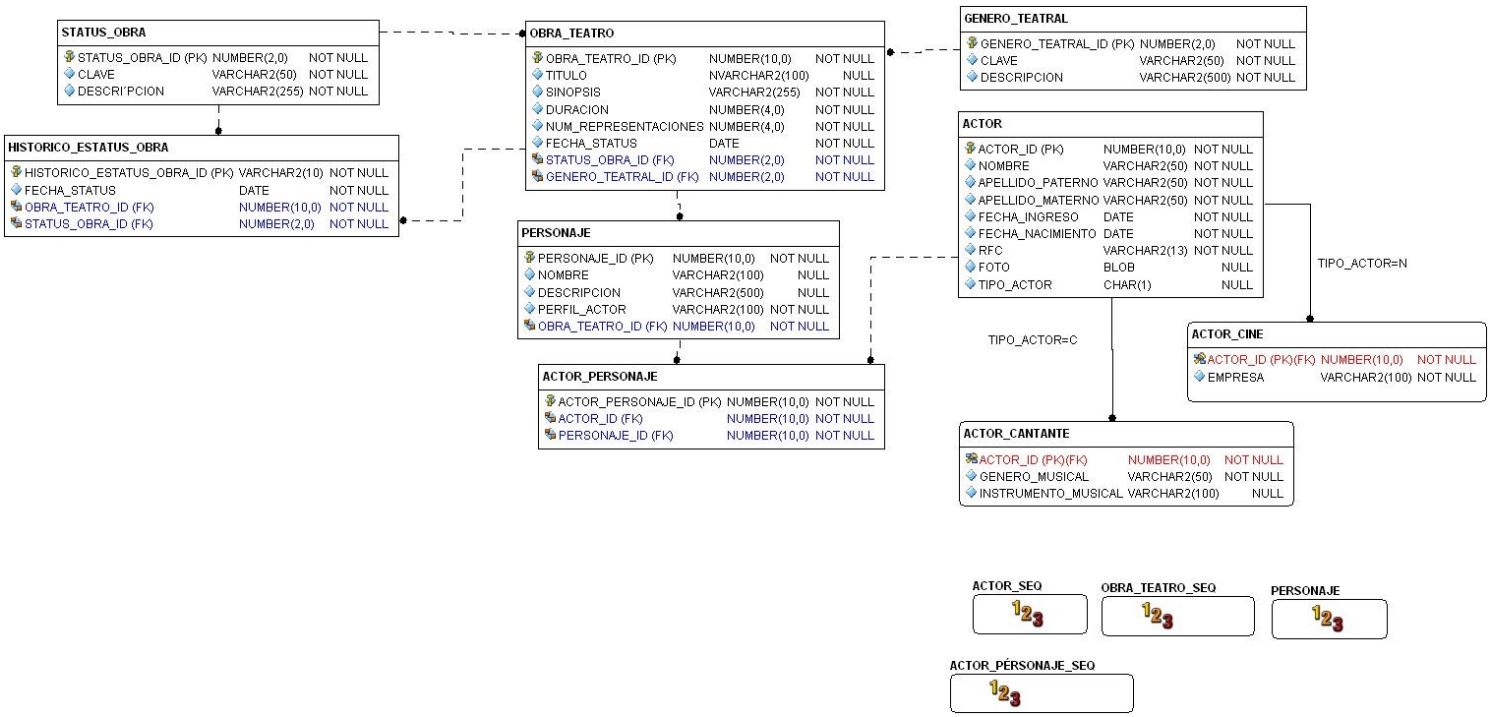
3. Estructura y código fuente del proceso de negocio.

A continuación se presenta parte del código fuente que implementa este proceso de negocio. Para efectos del proyecto, se deberán realizar las siguientes actividades:

- Revisar cuidadosamente este código fuente, y se deberá agregar el código Java indicado por los comentarios que inician con "TODO" (TO – DO , cosas por hacer).
- Agregar el código fuente de los 2 DAOs empleando JDBC para realizar las operaciones requeridas en la base de datos. Leer los comentarios de la interface para una mayor comprensión. Para generar los valores de las llaves primarias, emplear secuencias, o en su defecto columnas INDENTITY o AUTO INCREMENT.
- El reporte de este proyecto no se imprime, ni se envía por email. Se deberá mostrar al profesor su ejecución. Solo se entregará de manera impresa los siguientes elementos:
  - La carátula del proyecto final con los nombres de hasta 2 integrantes
  - Comentarios y/o experiencias al implementar el proceso y/o conclusiones (una sola por reporte).
- Se realizarán preguntas del código a cualquiera de los integrantes, por lo que ambos deberán tener el 100% de comprensión de lo realizado.

3.1. Modelo entidad - relación.

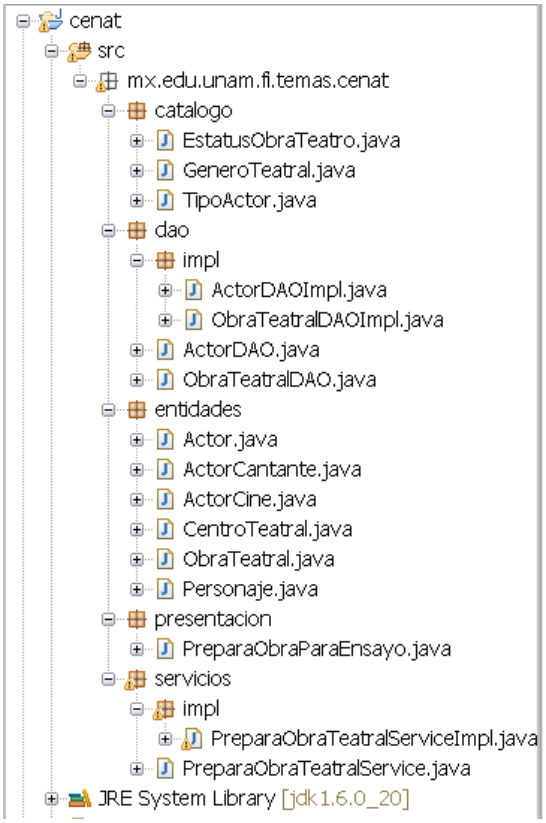
El siguiente diagrama muestra la estructura de la base de datos recomendada para implementar el proceso:



- Observar que para almacenar la historia de los status de la obra se emplea la tabla `HISTORICO_ESTATUS_OBRA`. Cada que cambia el status de la obra, se realiza un `update` en `OBRA_TEATRO` y un `insert` en el histórico.
- Para saber que actor interpretará a un personaje para una determinada obra se emplea la tabla `ACTOR_PERSONAJE`. Se genera un `insert` por cada asociación actor-personaje.
- Observar que para insertar o consultar los datos completos de un actor, se debe hacer join con las tablas `ACTOR_CANTANTE` y `ACTOR_CINE`, dependiendo el tipo de actor (`TIPO_ACTOR`).

3.2. Estructura del proyecto en Eclipse.

La siguiente imagen muestra la estructura del proyecto en eclipse y las clases desarrolladas para implementar este proceso de negocio:



- Observar, las entidades de negocio se guardan en el paquete entidades. Todas las clases solamente contienen los atributos de la entidad y sus métodos de acceso `get` y `set`.
- Las interfaces de negocio están en el paquete servicios, y sus implementaciones en el paquete `impl` (implementación).
- Las interfaces DAO están en el paquete `dao`, y sus implementaciones en el paquete `impl` (implementación).
- Observar que se cuentan con varios catálogos, incluido el catalogo de status de la obra de teatro representados por Enums (se revisa mas adelante).
- Finalmente, observar el paquete presentación. La clase `PreparaObraTeatro` se encarga de simular todas las clases y pantallas que se tendrían que aplicar para convertir este proyecto en un sistema web. La clase contiene un método `main` que es donde inicia la ejecución del proceso de negocio.

3.3. Código fuente del proceso de negocio.

3.3.1. *Ejemplo de entidades de negocio.*

Se omiten los métodos get y set.

```
public class ObraTeatral {

   private long obraTeatralID
   private String titulo;
   private String sinopsis;
   private int duracion;
   private int numRepresentaciones;
   private Date fechaEstatus;
   private EstatusObraTeatro estatus;
   private GeneroTeatral genero;
}
```

Clase Personaje:

```
public class Personaje {
   private long personajeID;
   private String nombre;
   private String descripcion;
   private String perfilActor;
   private Actor actorQueInterpreta;
}
```

Clase Actor:

```
public class Actor {

   private long actorID;
   private String nombre;
   private String apellidoPaterno;
   private String apellidoMaterno;
   private Date fechaNacimiento;
   private Date fechaIngreso;
   private String RFC;
   private char tipoActor;
   private byte[] foto;

}
```

3.3.2.*Ejemplo de catálogos.*

Observar el código del siguiente Enum, representa al catálogo de status de una obra teatral.

```
public enum EstatusObraTeatro {

   REGISTRADA(1,"REGISTRADA"),EN_PREPARACION(2,"EN PREPARACION"),LISTA_PARA_PUBLICAR(3,
   "LISTA PARA PUBLICAR"),EN_CARTELERA(4,"EN CARTELERA"),CONCLUIDA(5,"CONCLUIDA");

   private int id;

   private String clave;

   /**
     * Constructor privado
     * @param id
     * @param clave
     */
   private EstatusObraTeatro(int id,String clave) {
      this.id= id;
      this.clave= clave;
   }

   /**
     * Observar este método, se emplea para obtener una instancia
     * de esta clase  a partir de un id. El método values es heredado
     * enautomatico de la clase Enum.
     * @param id
     */
    public static EstatusObraTeatro valueOf(int id) {
       for (EstatusObraTeatro estatus :values()) {
         if (estatus.id== id) {
         return estatus;
       }
     }
      throw new IllegalArgumentException("el id especificado no es valido para obtener"
         +" una instancia de EstatusObraTeatro");
    }


   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id= id;
   }

   public String getClave() {
      return clave;
   }

   public void setClave(String clave) {
     this.clave= clave;
   }
}
```

3.3.3.*Código de la clase PreparaObraTeatro.*

Aquí inicia la ejecución de todo el proceso. Iniciar la revisión del código en esta clase.

```
/**
 * Aquí inicia la ejecución del proceso de negocio, Esta clase simula la parte web.
 * @author Jorge A. Rodríguez Campos (jorgerdc79@hotmail.com)
 * @version 1.0
 * @since 1.0
 */
public class PreparaObraParaEnsayo {

/**
   * Servicio empleado para invocar la ejecución del proceso de negocio. Ya en
   * sistemas en forma, se emplean otros mecanismos para crear instancias del
   * servicio.
   */
private PreparaObraTeatralService service=new PreparaObraTeatralServiceImpl();

/**
   * métodomain
   * @paramargs
   */
public static void main(String[] args) {
   new PreparaObraParaEnsayo().preparaObra();
}

/**
   * Este método se encarga de interactuar con el servicio de negocio, simula la
   * navegación por ejemplo, en un sistema web.
   */
private void preparaObra() {
   ObraTeatral obra;
   List<Actor>actoresDisponibles;

   List<Personaje> personajes;
   // en sistemas reales, los mensajes a la consola no se mandan con
   // System.out.println, se emplean otros mecanismos especializados en el
   // manejo de bitácoras. Ejemplos: Log4j, commons-logging, SLF4J. Para
   // propósitos del curso, se emplea System.
   System.out.println("Preparando una nueva obra de teatro para ensayo");

   // estos datos se recuperarían de una pagina web, aquí solo se simulan
   // asignando los datos de la obra y de sus personajes.
   obra=getObraFicticia();
   personajes=getPersonajesObra();
   System.out.println("Validando y registrando la nueva obra con sus personajes.");
   obra=service.registraObraTeatral(obra, personajes);

   System.out.println("La obra ha sido registrada");

   // aquí el sistema podría poner una pagina web para indicar que la obra se
   // registro exitosamente, y un botón de continuar para presentar una lista
   // con todos los actores existentes para asociarlos con su personaje.
   System.out.println("Continuando, buscando Actores disponibles para la obra");
   actoresDisponibles=service.consultaActoresExistentes();

   // Aquí el sistema le presentaría la lista de actores en la pantalla, y el
   // usuario seleccionaría por cada personaje al actor que lo representara.
   // Dentro de cada objeto Personaje hay una agregación con Actor para saber
   // que actor va a
   // representarlo. Este método simula esa selección.
   asignaActorAPersonaje(actoresDisponibles, personajes);

   // finalmente, invoca al servicio para que se prepare la obra para ensayo
   // con todos los datos seleccionados
   System.out.println("preparando la obra para ensayo");
   service.preparaObraParaEnsayo(obra, personajes);
}
```

```
/**
   * @param actoresDisponibles
   * @param personajes
   */
private void asignaActorAPersonaje(List<Actor>actoresDisponibles,List<Personaje> personajes) {
// Aqui se simularia la asignación de un actor a un personaje empleando el
// método actorQueInterpreta en la clase Personaje.
// TODO, PROGRAMAR ESTE METODO
  }

/**
   * @return
   */
private List<Personaje>getPersonajesObra() {
// Este método genera una lista ficticia de personajes de una obra de
// teatro.
// TODO, PROGRAMAR ESTE METODO
return null;
  }

/**
   * @return
   */
private static ObraTeatral getObraFicticia() {
// Este metodo genera un objeto ficiticio que representa a una obra de
// teatro.
// TODO, PROGRAMAR ESTE METODO
return null;
  }
}
```

Observar que en esta clase se invoca al servicio de la capa de negocio representado por el atributo de instancia `service`que es una instancia que implementa a la interface`PreparaObraTeatralService`.

3.3.4.*Código de la interface PreparaObraTeatralService*

```
/**
 * Este servicio de negocio define las operaciones requeridas para preparar una
 * obra de teatro, desde que se registra, hasta que queda lista para ponerse en
 * escena.
 * @author Jorge A. Rodríguez Campos (jorgerdc79@hotmail.com)
 * @version 1.0
 * @since 1.0
 */
public interface PreparaObraTeatralService {

/**
   * Este método se encarga de preparar una obra para ensayo:<br>
   * 1. Actualiza el estatus de la obra a EN PREPARACION <br>
   * 2. Registra su estatus en su histórico <br>
   * 3. Asocia al actor que representará a cada personaje.
   * @param obra
   * @param personajes
   */
void preparaObraParaEnsayo(ObraTeatral obra,List<Personaje> personajes);

/**
   * Actualiza el estatus de una obra teatra y agrega una entrada a su
   * historico.
   * @param obra
   */
void actualizaEstatusObra(ObraTeatral obra);

/**
   * Registra una nueva Obra teatral, le asigna el estatus REGISTRADA, le asigna
   * la fecha actual, le asigna su id, regustra su correspondiente historico de
   * estatus y finalmente registra los personajes de la obra.
   * @param obra
   * @param personajes
   * @return El mismo objeto pero con sus atributos actualizados. El metodo pudo
   *         haber sido marcado como void, ya que los atributos se actualizan
   *         por referencia y no es necesario regresar el mismo objeto.
   */
ObraTeatral registraObraTeatral(ObraTeatral obra,List<Personaje> personajes);
```

```
/**
   * Este metodo realiza una consulta en la base de datos de todos los actores
   * existentes que pueden ser seleccionados para representar a un personaje.
   * @return
   */
List<Actor>consultaActoresExistentes();

}
```

3.3.5.*Código de la implementación de la interface PreparaObraTeatralService*

```
/**
 * Este servicio procesa las peticiones de la capa de presentacion para preparar
 * una obra de teatro.
 * @author Jorge A. Rodríguez Campos (jorgerdc79@hotmail.com)
 * @version 1.0
 * @since 1.0
 */
public class PreparaObraTeatralServiceImpl implements PreparaObraTeatralService {

// Aqui se agregan 2 atributos de instancia que representan instancias de las
// clases de la capa de persistencia.
private final ActorDAO actorDAO=new ActorDAOImpl();

private final ObraTeatralDAO obraTeatralDAO=new ObraTeatralDAOImpl();

/*
   * La documentación de este método se encuentra en la clase o interface que lo
   * declara (non-Javadoc)
   * @seemx.edu.unam.fi.temas.cenat.servicios.PreparaObraTeatralService#
   * preparaObraParaEnsayo(mx.edu.unam.fi.temas.cenat.entidades.ObraTeatral,
   * java.util.List)
   */
@Override
public void preparaObraParaEnsayo(ObraTeatral obra,List<Personaje> personajes) {
// TODO PROGRAMAR ESTE METODO, REALIZAR LAS 3 ACCIONES QUE SE DESCRIBEN EN
// LA INTERFACE. PARA ASOCIAR LOS PERSONAJES CON SU AUTOR, usar el metodo
// asociaActorAPersonaje (revisar la interface y el modelo ER
// que se incluye en este documento).

  }

/*
   * La documentación de este método se encuentra en la clase o interface que lo
   * declara (non-Javadoc)
   * @seemx.edu.unam.fi.temas.cenat.servicios.PreparaObraTeatralService#
   * actualizaEstatusObra(mx.edu.unam.fi.temas.cenat.entidades.ObraTeatral)
   */
@Override
public void actualizaEstatusObra(ObraTeatral obra) {
   System.out.println("Actualizando estatus de la obra");
   obra.setFechaEstatus(new Date());
   obraTeatralDAO.actualizaEstatusObra(obra);
   System.out.println("Agregando una entrada al historico");
   obraTeatralDAO.registraEstatusEnHistorico(obra);
   System.out.println("Estatus de la obra actualizado.");

  }

/*
   * La documentación de este método se encuentra en la clase o interface que lo
   * declara (non-Javadoc)
   * @seemx.edu.unam.fi.temas.cenat.servicios.PreparaObraTeatralService#
   * registraObraTeatral(mx.edu.unam.fi.temas.cenat.entidades.ObraTeatral,
   * java.util.List)
   */
@Override
public ObraTeatral registraObraTeatral(ObraTeatral obra,List<Personaje> personajes) {
  // TODO PROGRAMAR ESTE METODO PARA QUE REALICE EL REGISTRO DE LA OBRA EN LA
  // BASE DE DATOS, TOMANDO EN CUENTA LA DOCUMENTACIÓN DE LA INTERFACE. EL EL
  // DAO ObraTeatralDAO, emplear el metodoregistraObraTeatral y
  // registraPersonajes.

   return null;
  }
```

```
/*
   * La documentación de este método se encuentra en la clase o interface que lo
   * declara (non-Javadoc)
   * @seemx.edu.unam.fi.temas.cenat.servicios.PreparaObraTeatralService#
   * consultaActoresExistentes()
   */
@Override
public List<Actor>consultaActoresExistentes() {
   // TODO INVOCAR LA CONSULTA EMPLEANDO EL DAO ActorDAO, emplear el metodo
   // consultaActoresExistentes, invocarlo 2 veces, la primera vez obtiene los
   // actores cantantes y la segunda vez a los de cine. Regresar una lista com
   // ambos grupos.
   return null;
  }

}
```

Observar que en esta clase se realiza la invocación de los DAOs los cuales se encuentran en la capa de persistencia, en este caso los daos `ActorDAO` y `ObraTeatralDAO`.

Finalmente, si se programan más procesos de negocio de la lista de procesos mencionada al inicio del documento, se pueden obtener hasta 2 puntos extras, siempre y cuando los procesos funcionen y estén implementados de forma correcta. Otra opción, puede ser, implementar este mismo proceso de negocio, pero integrando alguna pagina web, o empleando Hibernate.

El proyecto se puede realizar en equipos de máximo 2 personas.