Escuela de Ingeniería	
Examen de: Arquitectura de software	**Código de materia: 3851**
Fecha: 09-08-10	**Hoja 1 de 10**

Teórico 60 puntos

NOTA: Las respuestas que se brindan en este documento no tienen la completitud esperada en un examen. Son solamente con el fin de orientar sobre las posibles respuestas a cada pregunta

1. Definición de Arquitectura de Software

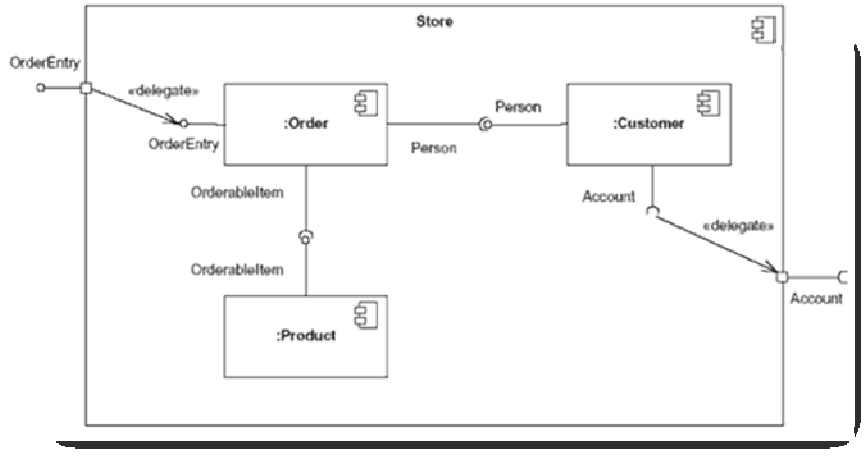
(7 puntos) Explique, detalladamente la definición de arquitectura de software de Bass.
RESPUESTA
Ver el Capítulo 2 del libro del curso, Sofware Architecture In Practice (Second ed.).

2. UML y Arquitectura

(15 puntos) En base a la siguiente descripción narrativa de un componente modelado en UML obtenga la representación gráfica.

- ***Descripción***: *El componente se denomina "Store" y provee una interface de nombre "OrderEntry" mientras que requiere de una interface "Account". La estructura interna del componente debe contemplar (o contener) otros tres componentes: "Order", "Customer" y "Product". Se emplean puertos para señalar explícitamente que la interface "OrderEntry" delega su procesamiento en la interface "OrderInterface" que expone el componente "Order". Para la interface requerida, es el componente "Customer" delega el procesamiento en el puerto "Account". Finalmente la interacción interna entre los tres componentes es la siguiente: El componente "Order" requiere de una interface que provee el componente "Product" de nombre "OrderItem" y requiere de una interface "Person" que provee el componente "Customer".*

RESPUESTA
En base a la narrativa, el componente quedaría modelado de la siguiente manera:



3. Patrones & Estilos de Arquitectura

(20 puntos) Se cuenta con la siguiente información para diseñar la arquitectura de un sistema de software.

i) Se sabe que una aplicación Web denominada *Sistema de Préstamos* estará compuesta por un número importante de componentes de diferentes niveles de abstracción (Presentación, Negocio, Acceso a Datos y Utilidades comunes a todos). De modo que es necesario estructurar la aplicación para soportar fundamentalmente la mantenibilidad.

ii) Se dispone de la descripción informal de un caso de uso:

> Caso de Uso: **Solicitar un Préstamo**.
> Paso 1: El usuario ingresa los detalles de solicitud de préstamo y acepta su envío.
> Paso 2: El Sistema de Préstamos envía un mensaje al Sistema Contable para obtener los datos históricos del cliente en su base de datos.
> Paso 3: El Sistema Contable, una vez que ha obtenido del repositorio los datos, envía un mensaje al Sistema de Préstamos con los datos.
> Paso 4: El Sistema de Préstamos recibe la información del Sistema Contable y procesa los datos para ejecutar una regla de negocio que le permite inferir si el préstamo es otorgado o no.

Para este mismo caso de uso se sabe además que tiene asociado la siguiente información de contexto: Las aplicaciones se integran exclusivamente mediante mensajes y tanto los que envían como los que reciben desconocen sus identidades (esto es, están fuertemente desacoplados).

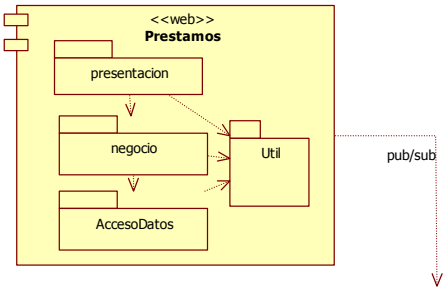
Se pide:

a) Identifique y describa los patrones & estilos arquitectónicos que emplearía para I) y II).

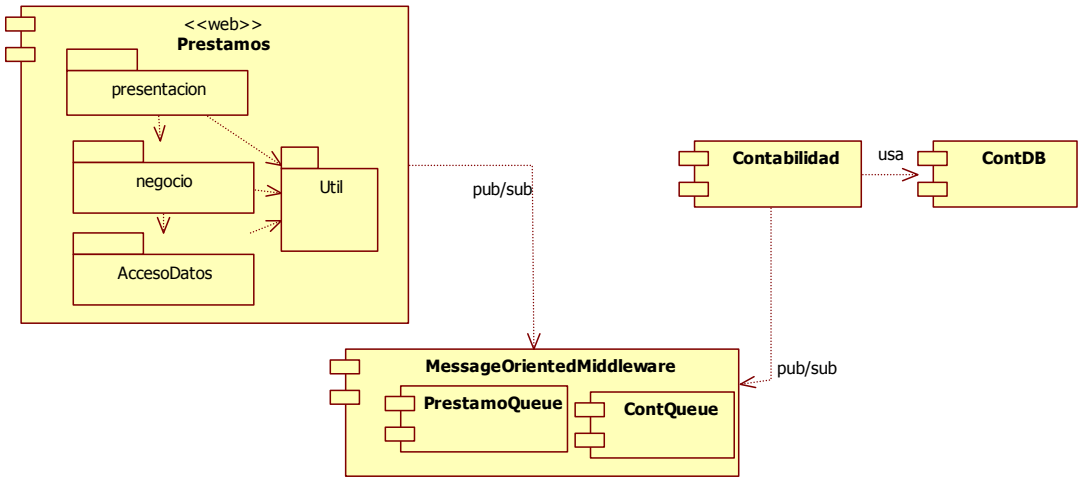
b) Bosqueje un modelo de componentes que refleje las interacciones entre los elementos de la solución.

RESPUESTA

a) El patrón / estilo arquitectónico que favorece principalmente mantenibilidad (entre otros atributos como testeabilidad) en la presencia de un número importante de componentes con distintos grados de abstracción es aquel que permite estructurar la aplicación / componente mediante capas (layering). En este caso se muestran las capas y sus relaciones en el componente de Prestamos.



- En el diagrama se muestran las capas y las relaciones entre ellas.
- Se emplea un **Publish/Subscribe** para desacoplar la comunicación entre los sistemas Prestamos / Contabilidad. Debido a que este componente tiene presencia en run-time se bosqueja como un componente. (Desde el punto de vista del modelado también es válido omitir el componente e indicar en las relaciones que se emplea msg/queue).
- Como se puede ver en la solución existen dos queues (será válido considerar también una sola). Una para enviar datos de Prestamos a Contabilidad y la otra para registrar la respuesta de Contabilidad a Prestamos.



4. Tácticas de Arquitectura

(12 puntos) Un arquitecto ha tomado las siguientes decisiones en cuanto al diseño de una aplicación:

1. Dado que todos los objetos de negocio tienen en común tanto estado como comportamiento, ha procedido a crear un nuevo paquete que contiene las entidades que permiten *generalizarlo*, definir constantes comunes y demás construcciones.

2. El sistema emplea una librería externa cuya API es susceptible de cambios que pueden impactar los módulos que la emplean. Para ello, el arquitecto ha provisto una mecánica que consiste en el empleo de un Adapter (o wrapper) para evitar las invocaciones directas puesto que el cambio de la API de la biblioteca ocasionaría *efectos en cascada* si cambia la sintaxis de los servicios.

3. Como método para reconocer las *faltas* que ocurren en la aplicación ha diseñado un mecanismo de gestión de Excepciones propias de la aplicación.

4. Sobre un conjunto de datos finitos, que no tienen porqué cambiar pueden almacenarse en un *caché* (por ejemplo, una lista en memoria) cuando el sistema es inicializado.

RESPUESTA

1. Modificabilidad: El arquitecto esta **localizando las modificaciones**. En particular está empleando una táctica para generalizar un modulo.

2. Modificabilidad: El arquitecto está **Previniendo los efectos en cascada** de los cambios ocasionados por la API de la librería externa.

3. Disponibilidad: Está empleando una técnica de **detección de faltas** mediante el manejo de Excepciones.

4. Performance: Está **gestionando los recursos** de datos, en particular está manteniendo copias de los datos mediante el empleo de caché.

5. Atributos de calidad y Escenarios

(6 puntos) Clasifique los siguientes escenarios en función del atributo de calidad al que corresponden indicando para cada uno: Fuente, Estimulo, Artefacto, Respuesta y Medida.

- *Cuando un usuario está realizando compras en línea el sistema debe evitar que el usuario realice por error compras duplicadas en el 100% de los casos*

RESPUESTA
Atributo: usabilidad
Fuente: Usuario
Estimulo: realizando compras en línea
Artefacto: sistema
Ambiente: en ejecución
Respuesta: debe evitar que el usuario realice por error compras duplicadas
Medida: en el 100% de los casos

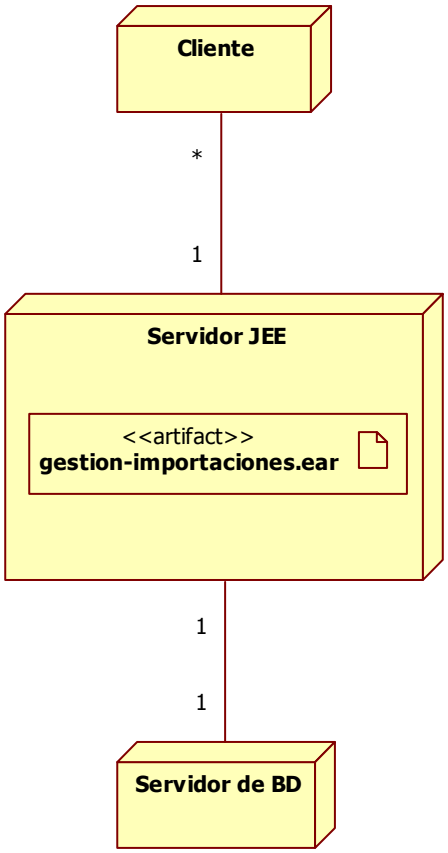
- *Cuando desarrollador, al finalizar el desarrollo un componente, realiza la prueba unitaria debe poder controlar los métodos que son invocados de forma de poder lograr una cobertura del 85% de los métodos dentro de cada ejecución de la prueba*

RESPUESTA
Atributo: Testeabilidad
Fuente: Desarrollador
Estimulo: realiza la prueba unitaria
Artefacto: componente
Ambiente: al finalizar el desarrollo de un componente
Respuesta: debe poder controlar los métodos que son invocados
Medida: lograr cobertura del 85% de los métodos dentro de cada ejecución de la prueba

Tecnología 40 puntos
NOTA IMPORTANTE: Responder en hoja aparte

Una empresa importadora de productos informáticos está desarrollando una aplicación para gestionar la mercadería que ingresa en cada importación. El arquitecto de software ha decidido que el sistema será implementado sobre la plataforma **Java Enterprise Edition (JEE)**, y ahora debe definir los aspectos arquitectónicos de la aplicación y las tecnologías que se usarán para resolver las diferentes necesidades.

La aplicación, llamada **Gestión de Importaciones**, se ejecutará en un servidor ubicado en la oficina central de la empresa, y será accedida por sucursales a lo largo de todo el país. Para la persistencia de datos, se utilizará una de base de datos relacional. El siguiente diagrama ilustra el despliegue de la aplicación como un archivo **EAR** en el servidor central:



A continuación, se detallan las necesidades para diferentes aspectos del sistema, y en cada uno de los casos será necesario determinar cómo resolver estos requerimientos.

6. Presentación

El único requerimiento clave que se menciona es el siguiente: dado que la funcionalidad del sistema será actualizada regularmente (debido a cambios frecuentes en las regulaciones aduaneras), surge la necesidad de **evitar** que las terminales en las sucursales que acceden al sistema se vean afectadas por estas actualizaciones. Es decir, no debería ser necesario instalar nuevo software cada vez que se despliega una nueva versión de la aplicación.

Se pide:

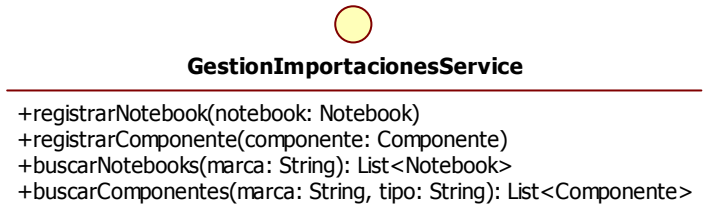
a) (6 puntos) Qué tipo de cliente JEE se usará en las terminales **(justifique su respuesta)**

b) (8 puntos) Describa el patrón/estilo que propone JEE para asegurar la separación entre la presentación y la lógica de la aplicación

RESPUESTA:
a) Para no afectar las terminales se aplica un cliente WEB de forma que todos los cambios estén centralizados en el servidor de aplicaciones.
b) El estilo es el de capas (TIERS). Ver Tutorial de JAVA EE. “Chapter 1: Overview”, página 42, "Distributed Multitiered Applications"

7. Negocio

De acuerdo a los requerimientos funcionales identificados para el sistema, el arquitecto ha definido la siguiente interfaz para la fachada que encapsula la lógica de negocio de la aplicación:



Adicionalmente, la empresa pretende que los distribuidores interesados puedan ser notificados cada vez que se registra un nuevo notebook en el sistema, para que éstos a su vez puedan informar a sus clientes de la llegada de un nuevo portátil al mercado.

Se pide:

a) (6 puntos) Qué tipo de componente JEE utilizaría para implementar la interfaz de fachada (Justifique su respuesta)

b) (8 puntos) Describa como resolvería el requerimiento de notificar a los distribuidores, detallando qué **recurso** administrado por el servidor de aplicaciones utilizaría para este tipo de notificación e indicando la **serie de pasos** que debería implementar para enviar la notificación (se puede usar pseudocódigo para explicar este procedimiento).

c) (6 puntos) Suponiendo que el sistema de un distribuidor también está basado en la plataforma JEE, ¿qué componente podría utilizar para recibir las notificaciones del importador?

RESPUESTA

a) El tipo de componente a utilizar es un EJB Session Bean porque asegura el manejo transaccional de las operaciones descriptas en la fachada. El mismo debería ser de tipo Stateless porque no es necesario mantener el estado conversacional con el cliente, cada invocación a una operación es única e independiente de las demás.
b) El recurso a utilizar es un Destination de tipo Tópico. Los distribuidores que quieren ser notificados se subscriben al mismo, mientras que el importador se encargaría de publicar las notificaciones. Los pasos necesarios para enviar una notificación se detallan en el Capitulo 31 "The JavaMessage Service API", página 902 del Tutorial de JAVA EE.
c) Un Message Driven Bean que será el encargado de subscribirse al Tópico.

8. Persistencia

A continuación se presentan las clases Java desarrolladas para modelar las entidades del dominio. Se decidió utilizar JPA para resolver el mecanismo de persistencia del modelo.

```
public class Fabricante {
    private String nombre;
    private String pais;
    // getters & setters
}
```

```
public class Notebook {
   private Fabricante fabricante;
   private String modelo;
   private List<Componente> componentes;
    // getters & setters
}
```

```
public class Componente {
    private Integer partNumber;
   private Date fechaFabricacion;
    private String marca;
    private Notebook notebook;
   // getters & setters
}
```

```
public class DiscoDuro extends Componente {
    private Integer capacidad;
    private Integer rpm;
   // getters & setters
}
```

```
public class CPU extends Componente {
   private Integer cantidadNucleos;
   private Integer frecuenciaReloj;
   // getters & setters
}
```

```
public class MemoriaRAM extends Componente {
    private Integer capacidad;
   private Integer frecuenciaReloj;
   // getters & setters
}
```

Se pide:

a) (6 puntos) Agregue las anotaciones JPA que entienda suficientes para asegurar la persistencia de todas las clases. La estrategia a utilizar con las herencias es SINGLE_TABLE

RESPUESTA

```
@Entity
@Table(name="T_FABRICANTES")
public class Fabricante {

   @Id
   @Column(name="NOMBRE")
   private String nombre;

   @Column(name="PAIS")
   private String pais;

   // getters & setters
}
```

```
@Entity
@Table(name="T_NOTEBOOKS")
public class Notebook {

   @ManyToOne
   @JoinColumn(name="FK_FABRICANTE")
   private Fabricante fabricante;

   @Id
   @Column(name="MODELO")
   private String modelo;

   @OneToMany(mappedBy="notebook")
   private List<Componente> componentes;

   // getters & setters
}
```

```
@Entity
@Table(name="T_COMPONENTES")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
```

```
@DiscriminatorColumn(name="REC_TYPE", discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="COMP")
public class Componente {

   @Id
   @Column(name="PART_NUMBER")
   private Integer partNumber;

   @Temporal(TemporalType.DATE)
   @Column(name="FCH_FABRICACION")
   private Date fechaFabricacion;

   @Column(name="MARCA")
   private String marca;

   @ManyToOne
   @JoinColumn(name="FK_NOTEBOOK")
   private Notebook notebook;

   // getters & setters
}
```

```
@Entity
@DiscriminatorValue(value="HDD")
public class DiscoDuro extends Componente {

   @Column(name="CAPACIDAD_HDD")
   private Integer capacidad;

   @Column(name="RPM")
   private Integer rpm;

   // getters & setters
}
```

```
@Entity
@DiscriminatorValue(value="CPU")
public class CPU extends Componente {

   @Column(name="CANT_NUCLEOS")
   private Integer cantidadNucleos;

   @Column(name="FRECUENCIA_CPU")
   private Integer frecuenciaReloj;

   // getters & setters
}
```

```
@Entity
@DiscriminatorValue(value="RAM")
public class MemoriaRAM extends Componente {

   @Column(name="CAPACIDAD_RAM")
   private Integer capacidad;

   @Column(name="FRECUENCIA_RAM")
   private Integer frecuenciaReloj;

   // getters & setters
}
```

Tablas generadas:

```
create table T_COMPONENTES (
   REC_TYPE varchar(31) not null,
   PART_NUMBER integer not null,
   FCH_FABRICACION date,
   MARCA varchar(255),
   CANT_NUCLEOS integer,
   FRECUENCIA_CPU integer,
   CAPACIDAD_HDD integer,
   RPM integer,
   CAPACIDAD_RAM integer,
   FRECUENCIA_RAM integer,
   FK_NOTEBOOK varchar(255),
   primary key (PART_NUMBER)
);

create table T_FABRICANTES (
   NOMBRE varchar(255) not null,
   PAIS varchar(255),
   primary key (NOMBRE)
);

create table T_NOTEBOOKS (
   MODELO varchar(255) not null,
   FK_FABRICANTE varchar(255),
   primary key (MODELO)
);
```