CURSO 2016 - 2017

ACTIVIDAD EVALUABLE Y CALIFICABLE

1. Portada con:

Asignatura: 71013035 – Diseño de Software

Año de la práctica:

Centro Asociado al que pertenece:

Tutor que le da asistencia:
(o grupo de tutoría Intercampus al que está adscrito)

Datos personales: Nombre y apellidos:

DNI o número de expediente:

Contacto (teléfono o correo electrónico):

Localidad de residencia:

Datos de coste: Horas de dedicación al estudio de los contenidos:

Nº de actividades no evaluables realizadas y horas de dedicación:

Horas de dedicación para realizar esta actividad:

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

swdesign@issi.uned.es

2. El enunciado y planteamiento del caso de estudio.

El dominio del problema es un **sistema de gestión para la venta de billetes de avión en una agencia de viajes** (e-Vuela).

El caso de estudio es similar al punto de venta (PdV) del libro de la asignatura y, como ahí, el trabajo se centrará, exclusivamente, en la lógica del negocio y en los servicios técnicos mínimos necesarios para implementar esa lógica en la aplicación. Es decir, aunque exista una capa de presentación, no debe tener en cuenta la interfaz de usuario, la presentación o cómo se capturan los eventos. Céntrese en la lógica de la interacción entre el usuario-agencia y su funcionamiento.

Se pretende que la aplicación tenga flexibilidad para ser utilizada tanto por un *operador* en un terminal de la propia agencia de viajes, por un *cliente* desde un dispositivo móvil o por un *administrador*, para realizar labores de supervisión y mantenimiento. Cada uno de estos perfiles determina las funcionalidades disponibles para un *usuario* concreto. Por tanto, la utilización de esta aplicación se restringe a usuarios registrados, con sus datos de acceso, personales y, en el caso de clientes, de facturación, almacenados en el sistema. En cualquier momento se puede registrar o dar de baja un nuevo *cliente* y cualquier usuario puede editar sus propios datos.

A parte de la interacción debida al mantenimiento de una cuenta (alta, baja, modificación de los datos del usuario, etc.), los servicios de la aplicación se originan cuando un usuario solicita o consulta un viaje en avión, con un número de plazas (pasajeros), entre un origen y un destino, para una fecha y condiciones de uso determinadas (necesidades especiales del viajero, facturación de equipaje, equipajes especiales, etc.). Con esos datos, la aplicación obtiene una *lista de itinerarios* mediante un sistema (externo) de planificación que, a su vez, consulta a las compañías aéreas que operen en cada punto de los itinerarios. Un itinerario consiste en una secuencia de uno o más vuelos entre el origen, las escalas intermedias (si las hay) y el lugar de destino. Cada itinerario contiene (referido a un único pasajero):

- Los datos de cada vuelo, con la siguiente información:
  1. Número de vuelo.
  2. Fecha y hora de salida (local del origen).
  3. Fecha y hora de llegada (local del destino).
  4. Aerolínea.
  5. Lugar de origen del vuelo.
  6. Lugar de destino del vuelo.
  7. Duración del vuelo.
- Duración total del viaje.
- Número de plazas (pasajeros).
- Coste total del viaje por pasajero. En el coste se incluyen los precios de cada vuelo así como las tasas (de aeropuerto, etc.) que correspondan.
- Condiciones de uso: facturación de equipaje incluido en el precio y otra información pertinente al respecto.

El resultado de la consulta (lista de itinerarios), se puede almacenar en su perfil, recuperar, actualizar y ordenar según la duración del viaje o su precio total. Con una lista de itinerarios, el usuario también puede seleccionar uno o más itinerarios y **reserva**rlos. Al hacer la reserva se '*bloquean*', durante un tiempo (24 h), las plazas solicitadas para los vuelos de ese itinerario (¡¡Aún no se compran!!). Cada reserva queda reflejada en la lista de itinerarios y, transcurrida su vigencia, desaparece (se '*libera*') al recuperar o actualizar la lista de itinerarios.

Tras hacer una reserva, si está vigente, un usuario puede **comprar** los billetes del viaje de la siguiente manera:

1. Registrando al *cliente*, especialmente sus datos de facturación; o editándolos o confirmándolos si ya estaba registrado.
2. Introduciendo los datos, necesarios para los vuelos, de cada viajero.
3. El *cliente* realiza el pago del importe completo: de todos los vuelos, de todas las tasas, de todos los viajeros...
4. La aplicación emite los billetes de todos los vuelos del itinerario, nominales para cada viajero.

Los detalles y simplificaciones admitidas son:

- Un usuario sólo puede manejar su propia información. Un *operador*, además, puede compartir las listas de itinerarios con cualquier *cliente* o emitir billetes. Un *administrador*, además, puede realizar cualquier operación y manejar la información de cualquier usuario con los otros dos perfiles.
- Se obviará la relación entre las compañías aéreas y la agencia de viajes, que capacita a esta última para '*bloquear*' un determinado número de pasajes (reserva) en un vuelo. No obstante, dichas compañías aéreas son las únicas que manejan cualquier elemento relacionado con los vuelos. Es decir, si bien el planificador (externo) será el encargado de obtener, por parte de las compañías, la información de los vuelos, serán ellas las únicas que emitan los billetes de sus vuelos, previo pago y nominales para cada viajero, cuando lo reclame la agencia.
- Las fechas y horarios son siempre locales. En el cómputo temporal se ignorarán desfases por usos horarios, cambio de año, etc.
- En la reserva y compra, no se contempla la opción de seleccionar asiento.
- Los precios se presentan y manejan en la divisa local de la geolocalización en la que opera el usuario. Aunque se ignorarán los procedimientos de conversión de divisa, la aplicación se debe poder usar en cualquier país.
- En el coste de los vuelos, se considerará el precio por la facturación de única maleta estándar por viajero estándar. Es decir, no se considerarán los sobrecostes por exceso de equipaje o condiciones especiales que queden fuera de la tarifa establecida en el itinerario. Así mismo, la comisión por los servicios (según las políticas de precios y descuentos de la agencia de viajes) estará incluida en el precio final del viaje.

- En la **compra**, el pago se realiza **a** la agencia de viajes. En la emisión de los billetes, se ignorará el procedimiento por el que la agencia, a su vez, reparte el pago de los distintos vuelos y tasas entre las compañías aéreas incluidas en el itinerario.
- Si la compra se realiza a través de un *operador*, el pago (del cliente a la agencia) podría ser en cualquiera de las modalidades (efectivo, tarjeta, etc.). Sin embargo, si es el *cliente* el que opera a través de un dispositivo móvil, sólo se considera el pago por medios electrónicos (tarjeta, portales de pago seguro, etc.)

3. **El enunciado de cada cuestión y las respuestas.** Para cada cuestión, incluirá los desarrollos, listados, diagramas y las argumentaciones que estime necesarios.

RECOMENDACIONES: Después de tratar de entender el planteamiento anterior del caso de estudio, léase todos los enunciados de las cuestiones siguientes, hasta el final. Esté atento a algunos indicios que le pueden ayudar para el enfoque de sus respuestas. Es posible que se le pida que describa o relacione algún elemento; lo cual podría hacerle sospechar que debería contar con él.

PREGUNTAS GUÍA: Estas preguntas no forman parte del ejercicio. Su objetivo es ayudar a que dirija sus conclusiones adecuadamente.

¿Qué hace, exactamente, este **software**?

¿De dónde provienen los estímulos? Es decir ¿quiénes son los actores y cuál debe ser su comportamiento (estímulos), frente al que debe reaccionar el software?

Una vez '*aislado*' el software y comprendido el objetivo primordial de su funcionamiento ¿qué secuencia de operaciones debe realizar, como reacción a los estímulos, para que el funcionamiento sea el deseado (Caso de Uso)?

En el comportamiento del software, la situación es parecida: un objeto reacciona ante un estímulo que proviene de otro objeto o del exterior (un actor). Esa es, precisamente, la tarea de asignar responsabilidades. Por tanto, la pregunta es: ¿qué objeto atiende un estímulo y cómo reacciona ante él? Nunca hay que perder de vista que, en definitiva, **estamos hablando de código** y que, esos estímulos, se traducen en llamadas a métodos. Por consiguiente, es inaceptable una invocación a un método inexistente o aquellas llamadas en las que se solicita una reacción que es imposible realizar porque no se dispone de la información necesaria (falta de parámetros, etc.) o ésta es inaccesible.

Sección 1. Evaluación de ***Casos de Uso***

1. (0'5 puntos) En relación al software de e-Vuela considerado en el caso de estudio, identifique al menos 4 casos de uso primarios y sus actores correspondientes. Represente los resultados en un diagrama de casos de uso de UML.

2. (1 punto) Escriba el caso de uso <<*ProcesarCompra*>> en un formato completo (se recomienda la variante '*en dos columnas*') y estilo esencial. Incluya tanto el escenario principal de éxito como 2 extensiones o flujos alternativos que pudieran ser frecuentes. Suponga que el escenario de partida para este caso de uso es el de un usuario registrado que ya está identificado en el sistema, que ya había realizado varias consultas para un viaje, para un mínimo de 2 viajeros, y las había almacenado en su perfil. Dicho usuario decide realizar la compra correspondiente a un itinerario, que ya había reservado, de una de las listas de itinerarios que ha recuperado de su perfil (el flujo básico de acciones comienza con la orden de compra del itinerario reservado y termina con la obtención de los billetes). En este flujo principal, considere sólo el pago por medios electrónicos, sin tener en cuenta las políticas de precios, descuentos o promociones de la agencia de viajes. No escriba un encabezamiento demasiado elaborado del caso de uso (es decir, omita *propósito*, *resumen*…); en su lugar, afronte directamente el transcurso típico de los acontecimientos.

Sección 2. Evaluación del ***Modelado Conceptual***

3. (2 puntos) En relación al caso de uso anterior <<*ProcesarCompra*>>, construya un Modelo de Dominio y represéntelo en notación UML. Represente los objetos conceptuales, las asociaciones y los atributos.

Vale la pena que dedique esfuerzo a elaborar el Modelo de Dominio, porque son los cimientos del desarrollo posterior. Es como un diagrama de clases pero sin métodos, sólo los atributos y las relaciones. En este diagrama debe verse con claridad la ejecución de la evolución de los acontecimientos que se ha descrito en el punto anterior. Se crean los objetos y algunas relaciones para reflejar dónde se realizan las acciones, y en qué secuencia, indicadas en la escritura del caso de uso. Pero dichas acciones utilizan unos datos o requieren determinada información y, ahora, es el momento de pensar y tomar algunas decisiones sobre dónde (en qué objetos) reside esa información: los atributos de cada objeto conceptual son la información que sólo él puede manejar.

Con esto se completa el mapa de la funcionalidad del caso de uso, del '*negocio*'.

Sección 3. Evaluación de los ***Eventos del Caso de Uso***

Antes de comenzar a asignar responsabilidades (diseño), se suele complementar la comprensión del comportamiento deseado para el sistema (como en el Modelo de Dominio que se acaba de elaborar) con la evaluación de los Eventos del Sistema. Para ello se construye un Diagrama de Secuencia del Sistema (DSS) en el que se analizan los eventos o estímulos que se originan en los actores y cómo reacciona el sistema. Es decir, cuál es el comportamiento del sistema software (tomándolo como un único objeto o elemento), en cuanto a su salida (la reacción '*externa*' hacia el actor), cuando recibe un evento (estímulo) de algún actor.

4. (1'5 puntos) Circunscrito al caso de uso anterior <<*ProcesarCompra*>>, construya un Diagrama de Secuencia (diagrama de interacción DS) en UML. Represente los actores y los eventos de los componentes del sistema para este caso de uso.

   NOTA: se pide un diagrama de secuencia en el que represente el paso de mensajes entre los actores y los objetos, **NO** del Sistema (DSS). Por tanto, represente las líneas de tiempo de los objetos identificados en el modelo en lugar de la del *sistema global*.

   A partir de este DS, especifique los contratos de **dos** de las siguientes operaciones: 'IniciarCompra' (inicio de la compra del itinerario reservado), '*AgregarViajero*' (agregar la información de cada viajero para poder emitir los billetes), 'RealizarPago' (pago con tarjeta de todo el viaje, con todos los billetes) o '*ImprimirBilletes*' (emisión e impresión de todos los billetes, de todos los vuelos y de todos los viajeros, tras comprobar el pago).

ATENCIÓN: a menudo, en otros ejercicios y exámenes, las operaciones como '*AgregarViajero*' no aparecen de forma explícita, si no con corchetes: <<*operación A*>>. Esa notación es un ejemplo, usted lo debería sustituir por los nombres que les haya puesto a las operaciones **principales** que haya elegido.

NOTA: se pide un diagrama de secuencia en el que represente el paso de mensajes entre los actores y los objetos, **NO** del Sistema (DSS). Por tanto, represente las líneas de tiempo de los objetos identificados en el modelo en lugar de la del *sistema global*.

NOTA: la segunda parte de esta pregunta (escritura de los contratos de las operaciones) es fundamental para realizar las preguntas de la Sección 4; puesto que prácticamente se aproxima a escribir el pseudocódigo del método (operación). Dicho de otra forma: facilita enormemente el desglose de los mensajes, los parámetros que necesitan, los valores que devuelven y las acciones que realizan.

Sección 4. Evaluación de la ***Asignación de Responsabilidades*** y ***Diseño de Colaboraciones***

5. (2 puntos) A partir del contrato de la operación <<*se omite la operación A*>> que haya indicado en el punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.

> ATENCIÓN: lo que hay entre corchetes <<*se omite...*>> es un ejemplo, usted lo debe sustituir por el nombre que le haya puesto a **la primera operación principal** que haya elegido en la pregunta anterior 4.

6. (2 puntos) A partir del contrato de la operación <<*se omite la operación B*>> que haya indicado en el punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.

> ATENCIÓN: lo que hay entre corchetes <<*se omite...*>> es un ejemplo, usted lo debe sustituir por el nombre que le haya puesto a **la segunda operación principal** que haya elegido en la pregunta 4.

Sección 5. Evaluación de los ***Diagramas de Clases*** de diseño

7. (0'5 puntos) Elabore un diagrama de clases para el caso de uso que se está tratando <<*ProcesarCompra*>> (DCD), centrado en la clase cuya responsabilidad es el registro de la compra de la reserva seleccionada y de la emisión de billetes, según se ha descrito en el caso de uso. Represente los nombres de todos sus atributos, asociaciones (con la navegabilidad) y métodos.

Sección 6. Evaluación de la ***Transformación del Diseño en Código***

8. (0'5 puntos) A partir de los anteriores diagramas de clases y colaboraciones, elabore y defina la clase que haya definido, en el desarrollo anterior, como responsable de la compra de la reserva seleccionada y de la emisión de billetes en el caso de uso <<*ProcesarCompra*>>. Incluya las definiciones de todas las variables que la componen (miembros), pero escriba solamente la definición completa del cuerpo para el método (o métodos) principal/es o más significativo/s: <<*se omite el método*>>. Ignore los pequeños detalles de sintaxis -el objetivo es evaluar la capacidad fundamental para transformar el diseño en código-. Utilice la sintaxis de Java.

> ATENCIÓN: lo que hay entre corchetes <<*se omite…*>> es un ejemplo, usted lo debe sustituir por los nombres que le haya puesto a los métodos que haya elegido como principales (por ejemplo, '*AgregarViajero*' e '*ImprimirBilletes*').

Por experiencias anteriores, a veces parece que estudiante tenga notables dificultades para llegar a cumplir, razonablemente, el objetivo final: escribir uno o dos métodos que realicen la operación que se le pide.

Estoy convencido de que no es por falta de capacitación; puesto que, el lector, ha recibido formación suficiente para poder escribir ese código sin grandes esfuerzos. Posiblemente se deba a que, su falta de experiencia, le lleva a incurrir en la siguiente paradoja: una metodología de desarrollo (el Proceso Unificado y el ciclo de vida iterativo) orientada a la comprensión rápida de los requisitos fundamentales y al avance rápido en el desarrollo (cercano al ágil), induce un bloqueo en la comprensión de esos requisitos funcionales y la crispación por aplicar los principios GRASP o encontrar patrones útiles conduce al fracaso. Parece que la inseguridad y el desconocimiento restringen el desarrollo con una rigidez tal que lo convierten en un modelo 'en cascada' y hacen zozobrar el resultado.

Si ocurre esto, mi recomendación es que utilice un '*salvavidas*'. Quizás, si '*quema sus naves*', recupere la visión del contenido y el alcance de la asignatura: buscar lo esencial para aprovecharlo. Y pueda valorar así la potencia de las herramientas que se le proponen.

Así que olvídese de los patrones, de la refactorización, de los principios GRASP, del ciclo de desarrollo… incluso de los objetos y OOP. Afronte el problema: ¿qué le piden? ¿una función que registre una compra? Pues escríbala. Si quiere, como en el libro. No tenga miedo, váyase al final del proceso; a ver si, así, consigue la perspectiva adecuada.

El método que acaba de escribir pertenecerá a una clase, necesitará unos datos de entrada y calculará otros. Algunos de esos datos no los provee, directamente, el actor que ha iniciado esta operación; sino que se obtendrán de alguna parte… Puede que no se haya dado cuenta, pero está usando el método de Descomposición Funcional Descendente y el Refinamiento Progresivo que estudió en la asignatura de "*Introducción a la Ingeniería de Software*" y que, seguramente, lleva utilizando toda la vida. Si se siente cómodo así, continúe 'marcha atrás'; asignando responsabilidades para esos cálculos y operaciones intermedias que está deduciendo. Ahora también puede aplicar los principios GRASP para esas asignaciones… Prosiga hacia '*atrás*' hasta llegar a un modelo de comportamiento: el Modelo de Dominio. ¡Ya está! ¡Ya lo ha hecho!

Bueno… está más cerca de conseguirlo.

Sobre este desarrollo tan simple (que, sin embargo, le dará confianza en sí mismo), es deseable incorporar ciertos aspectos cualitativos; como la independencia funcional, la ocultación o cierta flexibilidad. El avance realizado es más grande de lo que cree porque, si ya ha intentado aplicar alguno de los principios GRASP, habrá comprendido algunas de las carencias que posee su desarrollo inicial y, sobre todo, cómo quiere que se comporte el software que está construyendo (en este caso, el que le plantea el enunciado). En realidad, los 5 primeros principios GRASP, son una reutilización de los 'Conceptos Básicos' del diseño que ya ha estudiado en la asignatura de "*Introducción a la Ingeniería de Software*". Evidentemente, al aplicar estos principios (y los conceptos que les acompañan), se introducen modificaciones cualitativas en el desarrollo que tenía que le obligarán a modificar su diseño, el diagrama de clases, el código y, muy posiblemente, tendrá que reflejar ese cambio en el comportamiento de su sistema, en el Modelo de Dominio. En este momento es cuando está entrando en el significado y la utilidad de esta asignatura.

Pero aún hace falta adentrase un poco más. El problema plantea cuestiones adicionales (que suelen corresponderse con la flexibilidad del diseño – código) como el manejo de las listas de itinerarios, de dónde se obtienen las plazas de avión, el precio de los vuelos, el de las tasas de aeropuerto, la aplicación de cargos y comisiones diversos o la posibilidad de consultas y peticiones a elementos software que no sólo están fuera del sistema de estudio (la aplicación de la agencia de viajes), sino fuera del sistema de la empresa en la que se aloja y, por lo tanto, están totalmente indeterminadas la forma de comunicarse con ellos y de intercambiarse información (por ejemplo, las consultas a las líneas aéreas, reserva de plazas, precio de los billetes o emisión de los billetes por su parte). Estos nuevos requisitos son mejoras en el producto, requieren nuevas soluciones y los otros principios GRASP le servirán de guía para encontrarlas. A su vez, estas soluciones le obligarán a modificar el código, refinar el diseño, ajustar el Modelo de Dominio y reescribir el Caso de Uso.

Esta es la principal dificultad de la asignatura: el mantener ese doble flujo bidireccional y deductivo entre el planteamiento del problema completo, la construcción, paso a paso, de la solución hacia el código final y viceversa, desde el germen simplificado de la solución, incorporar requisitos adicionales hasta llegar a recoger el planteamiento completo del problema. Es como la dificultad en la coordinación del movimiento de la mano izquierda, que acompaña en

el piano, y la mano derecha, que lleva la melodía.

El problema está en que, sin experiencia, no se posee la fluidez que permite construir la solución final (o incorporar una modificación en ella), '*en paralelo*' a cada requisito que se evalúa. La agilidad y flexibilidad del proceso iterativo es porque se hace así y, si no lo hace, si intenta comprender todos los requisitos funcionales e incorporarlos en el Modelo de Dominio, sin considerar (simultáneamente) cómo afectan al Diagrama de Clases y al código final o si, mientras adquiere esa experiencia, se resiste a las '*idas y venidas*' (la *esencia* del iterativo), a utilizar mucho papel y emborronar sus diagramas, modificándolos constantemente; está utilizando la metodología opuesta: el rígido desarrollo en cascada en el que, si no ha completado el Modelo de Dominio, especificado toda la funcionalidad y comprendido totalmente el comportamiento deseado; no pasa a la siguiente fase. Esto produce múltiples indeterminaciones que desembocan en lo que, en computación, se podría llamar inter-bloqueo ('dead-lock' o 'mutual-exclusion'). Un fracaso.

La mejor manera de conseguir experiencia es trabajar como si no la tuviera… y aprender. Eso sí: no dude de su capacidad ni de la utilidad de lo que ya ha aprendido. Lo que se le pide no es difícil. Otra cosa es la aplicación de patrones de diseño para resolver algunos de estos problemas, la refactorización de la solución u otras cuestiones que se tratan en el libro; para las que sí es necesaria cierta experiencia.

Asuma que el objetivo mínimo de la asignatura es que adquiera esa experiencia, que aprenda a utilizar los principios GRASP para desarrollar software flexible y de una calidad aceptable. La recomendación es que tome esta actividad, las de años anteriores (o la parte que le interese), el ejemplo PdV del libro o cualquier otro problema que encuentre, para practicar con esas 'idas y venidas', la elaboración de los diagramas (en papel, no pierda el tiempo con la herramienta gráfica, sino en practicar y comprender), para adquirir conocimiento y experiencia y evitar hacer ese '*re-trabajo*' en el examen.

Otra sugerencia, si quiere ampliar su práctica, es que afronte (aunque sólo sea intelectualmente) otros casos de uso de este mismo escenario: 'Mantenimiento de cuentas de usuario', el manejo de las 'Consultas' o la gestión de perfiles de usuario (almacenamiento, recuperación o actualización de las listas de itinerarios obtenidas en una consulta). Es más, es muy posible que, antes de hacer el Modelo de Dominio de este caso de uso, le resulte útil pensar y elaborar un pequeño boceto de cómo se hace una consulta y se obtiene una 'lista de itinerarios'.