CURSO 2015 - 2016

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 **subsistema de transacciones simples de una entidad bancaria con sus clientes** (BankBox).

Restricciones y simplificaciones:

- Los clientes son, únicamente, personas físicas. Sólo se consideran los usuarios que son clientes de la entidad; es decir, son titulares de algún producto (cuenta, tarjeta, etc.) El acceso a los servicios requiere la identificación del cliente.
- Aunque las operaciones se podrían realizar a través de una interfaz Web, en cualquier dispositivo informático con esa capacidad, el estudio se reduce a los **cajeros automáticos** de la entidad.
- Los servicios que se van a considerar son:
    - Consultar saldo.
    - Información de movimientos en cuenta.
    - Obtener dinero en metálico.
    - Ingresar dinero en cuenta.
    - Ingresar dinero en tarjeta-monedero.
    - Transferencia entre dos cuentas (la de destino puede ser de otra entidad). La transferencia se realizará en la divisa local del cajero.

En cuanto al alcance del sistema, se pretende que permita operaciones:

- Con distintos tipos de productos (cuentas corrientes, de crédito, de ahorro, etc.); a los que se aplican distintos tipos de comisiones.
- Con distintos tipos de tarjetas (de crédito, de compras, de pago aplazado, monedero, etc.); a las que también se aplican diferentes cargos, según el tipo de tarjeta y la operación realizada.

Supóngase que la arquitectura de este subsistema tiene una estructura en 3 capas:

- Capa de presentación.
- Capa de la lógica del negocio.
- Capa del acceso a datos o de los servicios técnicos.

En adelante, en el escenario descrito, el estudio se hará sobre la capa de la lógica y los servicios del negocio. Es decir, aunque exista una capa de presentación, no debe tener en cuenta la interfaz de usuario, la presentación o la captura de eventos. Céntrese en la lógica de la interacción cliente-banco y su funcionamiento.

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 a las operaciones en el cajero automático, 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 <<*Transferencia*>> en un estilo y formato completo, esencial y breve. Incluya tanto el escenario principal de éxito (flujo básico correspondiente a que el cliente –**ya identificado y autorizado**— seleccione la operación, indique los datos necesarios, reciba la autorización –si es pertinente– y el comprobante) como 2 extensiones o flujos alternativos que pudieran ser frecuentes. 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 <<*Transferencia*>>, construya un Modelo de Dominio y represéntelo en notación UML. Represente los objetos conceptuales, las asociaciones y los atributos.

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.

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

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

   A partir de este DS, especifique los contratos de **dos** operaciones principales: <<*operación A*>> y <<*operación B*>>.

   ATENCIÓN: lo que hay entre corchetes <<*operación…*>> es un ejemplo, usted lo debe 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 operación principal que haya elegido en la pregunta anterior.

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 <<*Transferencia*>> (DCD), centrado en la clase cuya responsabilidad es registrar los datos de esa operación. 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 registrar los datos de la operación <<*TransferirDinero*>>. 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 o más significativo: <<*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 el nombre que le haya puesto al método principal que haya elegido.

Por experiencias anteriores, a veces parece que estudiante tenga notables dificultades para llegar a cumplir, razonablemente, el objetivo final: escribir un método que realice 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 transferencia de dinero? Pues escríbala. 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 uso de diferentes productos (cuentas) y tarjetas, 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 (el software **del** cajero), sino fuera del sistema de la entidad bancaria en la que se aloja y, por lo tanto, están totalmente indeterminadas la forma de comunicarse con ellos y de intercambiarse información (por ejemplo, la consulta de la cuenta de destino en la transferencia a una entidad externa o una solicitud de autorización a dicha entidad). 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.