Examen 23 de Julio 2010

Presentar la resolución del examen:

· Indique su nombre completo y número de cédula en cada hoja.
· Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.
· Escriba las hojas de un solo lado.
· **Comience cada ejercicio en una hoja nueva.**
· El examen es individual y sin material. **APAGUE SU CELULAR.**
· **Escriba con lápiz y de forma prolija.**
· Duración: 3 horas.

Problema 1 (25 puntos)

a) Responda de forma breve y concisa las siguientes preguntas:
- i) ¿Qué es una agregación compuesta?
- ii) ¿Qué es un tipo asociativo?
- iii) ¿Cuándo es necesario utilizar un rol?

b) Una compañía de desarrollo de software desarrollará un sistema de gestión de inventario. El sistema será desarrollado siguiendo una metodología iterativa e incremental. En cada una de sus 3 fases se irán agregando nuevas funcionalidades al sistema. Como analista de requerimientos se le pide a usted que realice el modelado del dominio del sistema.

i) Fase 1: Cuentas de inventario.

"Una cuenta de inventario se identifica por un nombre y registra la cantidad de unidades existentes (balance). Además, cada cuenta registra su secuencia de movimientos. Cada movimiento de una cuenta se identifica por un número (relativo a la cuenta a la que pertenece), contiene una breve descripción del mismo y la cantidad de unidades que entra o sale. El balance de una cuenta debe corresponderse con sus movimientos."

Se pide: Realice el modelo de dominio para esta fase y escriba en lenguaje natural sus restricciones.

ii) Fase 2: Transacciones de inventario.

"Una transacción registra un conjunto no vacío de movimientos de distintas cuentas. Todos los movimientos son generados por una transacción. Éstas se identifican por un número único en todo el sistema y solamente incluyen movimientos del mismo tipo (todos de entrada o todos de salida)."
Por ejemplo, una transacción es: "entran 5 unidades a la cuenta X, 6 a la cuenta Y y 3 a la cuenta Z".

Se pide: Realice el modelo de dominio para esta fase y escriba en lenguaje natural sus restricciones.

iii) Fase 3: Cuentas agregadas.

"Para facilitar la administración, el sistema debe permitir la agrupación de 2 o más cuentas en un nuevo tipo de cuenta llamada cuenta agregada. Las cuentas agregadas tienen un nombre y su balance es la suma de los balances de las cuentas que la componen. Las cuentas agregadas no tienen movimientos. Una cuenta agregada no puede agregarse a si misma."

Se pide: Realice el modelo de dominio para esta fase y escriba en lenguaje natural sus restricciones.

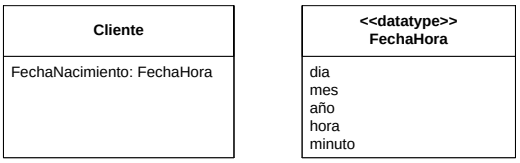
Nota: Todos los modelos deben ser independientes y solo deben agregar o cambiar lo necesario.

Problema 1 (25 puntos) – Solución

a)

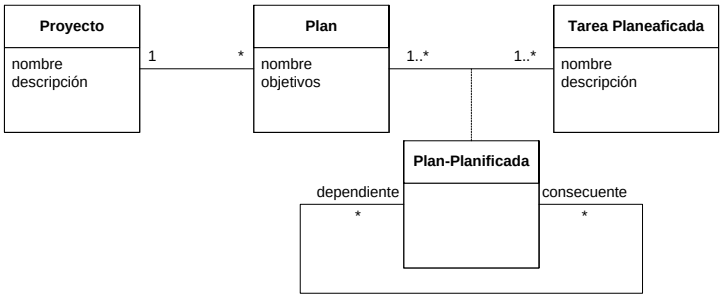
i. Ver slides de teórico: 09 – Análisis – Modelado de dominio slides 10 - 13.

ii.



b)

i.



Restricciones:

```
context Proyecto
-- El nombre identifica el proyecto.
inv: Proyecto.allInstances()->isUnique(nombre)


context Plan
-- El nombre identifica al plan.
inv: Plan.allInstances()->isUnique(nombre)


context Tarea
-- El nombre identifica la tarea.
inv: Tarea.allInstances()->isUnique(nombre)
```

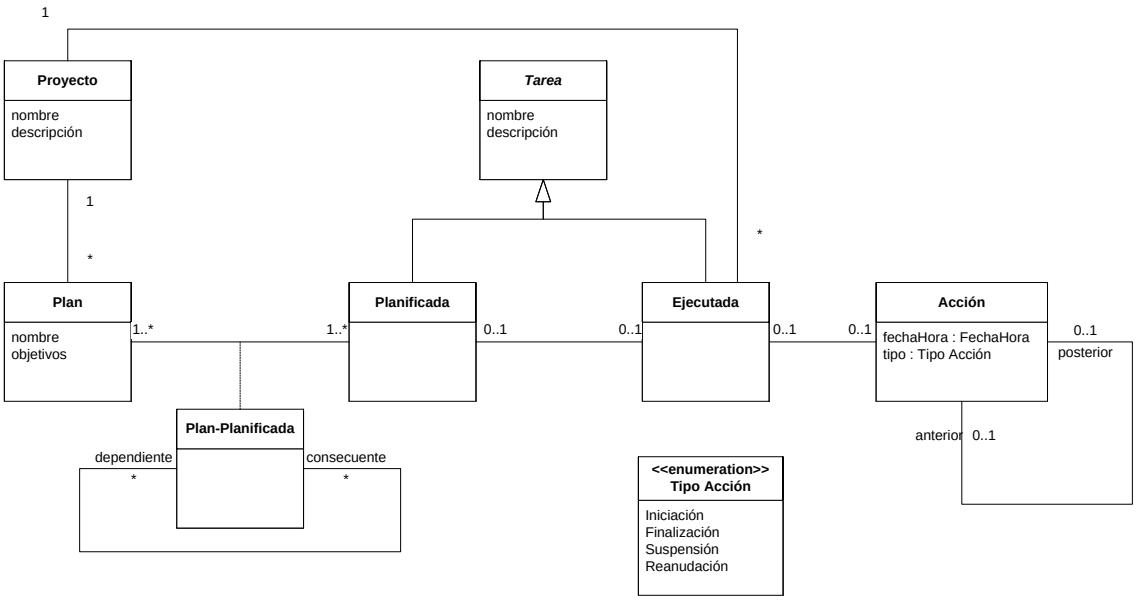
```
context Plan-Planificada
-- Las tareas de un plan sólo pueden estar relacionadas con otras
-- tareas del mismo plan.
inv: self.consecuente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Una tarea no puede depender de sí misma
inv: self.consecuente->excludes(self)


-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->excludes(self)
```

ii.



Restricciones:

```
context Proyecto
-- El nombre identifica el proyecto.
inv: Proyecto.allInstances()->isUnique(nombre)


context Plan
-- El nombre identifica al plan.
inv: Plan.allInstances()->isUnique(nombre)
```

```
context Tarea
-- El nombre identifica la tarea.
inv: Tarea.allInstances()->isUnique(nombre)


context Plan-Planificada
-- Las tareas de un plan sólo pueden estar relacionadas con otras
-- tareas del mismo plan.
inv: self.consecuente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Una tarea no puede depender de sí misma
inv: self.consecuente->excludes(self)


-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->excludes(self)


context Acción
-- Para cada acción, la fecha de la acción posterior debe ser mayor o igual
inv: self.posterior.fecha >= self.fecha

-- Una tarea iniciada puede ser suspendida o finalizada
inv: self.tipo = TipoAcción::Iniciación implies (self.posterior.tipo =
TipoAcción::Finalización or self.posterior.tipo = TipoAcción::Suspensión)

-- Una tarea suspendida sólo puede ser reanudada
inv: self.tipo = TipoAcción::Suspensión implies self.posterior.tipo =
TipoAcción::Reanudación


-- Una tarea finalizada no puede sufrir acciones posteriores
inv: self.tipo = TipoAcción::Suspensión implies self.posterior->isEmpty()
```

Problema 2 (25 puntos)

a) Conteste brevemente acerca de contratos:
   i. Qué cosas se especifican en las precondiciones.
   ii. Qué cosas se especifican en las postcondiciones.

b) Se desea desarrollar un sistema que permita a clientes consultar y comprar artículos de un catálogo on-line mediante la utilización de un carrito de compras. Cada artículo posee un identificador, una descripción y un precio. Es imprescindible que para poder confirmar la compra on-line, el cliente esté autenticado (es decir loggeado) en el sistema, no así para poder consultar artículos ni para agregarlos al carrito de compras.

El caso de uso principal es el siguiente:

Nombre: Compra on-line	**Actores: Cliente**
Descripción	El caso de uso comienza cuando el cliente inicia una nueva compra en el sistema. Éste mostrará una lista con todos los identificadores de artículos. El cliente podrá consultar un artículo por vez a partir de su identificador y el sistema le mostrará todos los detalles de ese artículo (identificador, descripción y precio). Luego de ver estos detalles, el cliente decide si agregar el artículo consultado al carrito de compras (junto con la cantidad de ese artículo a comprar) o no. Después de agregar al carrito todos los artículos de su interés, el cliente debe indicar la terminación de la compra. Si ya se encontraba loggeado en el sistema, se continuará con el proceso de compra on-line, de lo contrario el sistema le pedirá el ingreso de usuario y contraseña. Se le darán todas las oportunidades que sean necesarias para que se autentique, y no se permitirá continuar hasta que esté autenticado, pudiendo terminar inmediatamente el caso de uso si decide no loggearse. Una vez que se tiene autenticado al cliente (ya sea porque ya estaba loggeado o porque lo acaba de hacer) el cliente debe confirmar la compra, con lo cual el sistema controlará el stock de todos los artículos agregados al carrito. En caso de que no exista stock suficiente para algún producto, se le notifica al cliente y éste se des-loggea del sistema, no pudiendo modificar la compra. Si existe suficiente stock, el cliente ingresará los datos de su tarjeta de crédito (compañía, vencimiento y número) y se culminará el proceso de compra on-line.

Realice un único Diagrama de Secuencia del Sistema para el caso de uso anterior, incluyendo toda la información contenida en el mismo.

Problema 2 (25 puntos) - Solución

a) Las precondiciones especifican:

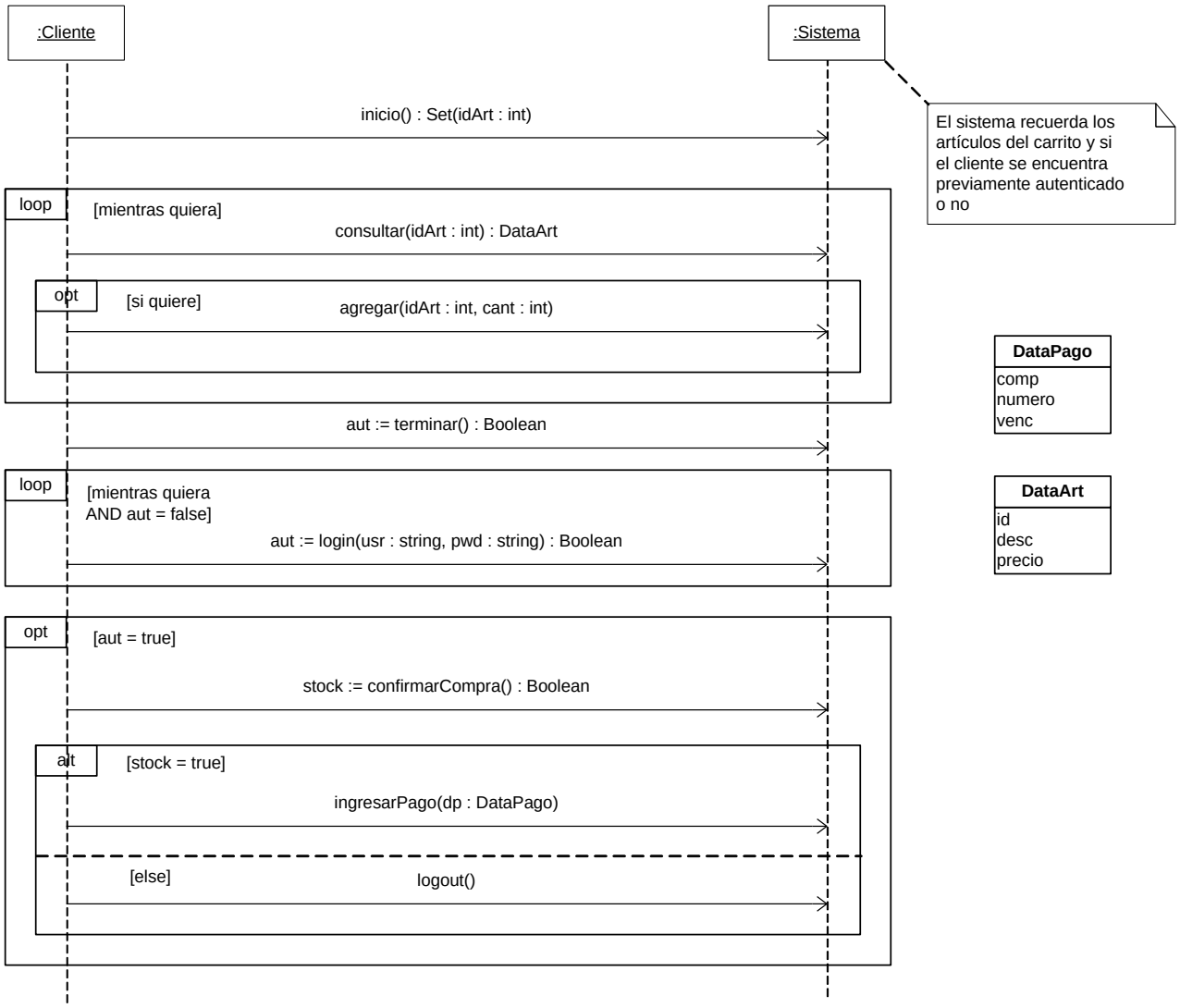
- Los valores de los parámetros de la operación
- El estado del sistema antes de ejecutar la operación, en términos de:
  - Que un objeto existe
  - Que un objeto no existe
  - Que un link existe
  - Que un link no existe
  - Propiedades sobre valores de atributos de objetos

Las postcondiciones especifican:

- El valor de retorno de la operación
- El estado del sistema luego de ejecutar la operación, en términos de:
  - Que un objeto existe
  - Que un objeto no existe
  - Que un link existe
  - Que un link no existe

b)

Parte i:



Nota: se podría representar también el logout del cliente luego de ingresar el pago.

Problema 3 (25 puntos)

a) En forma breve y concisa:

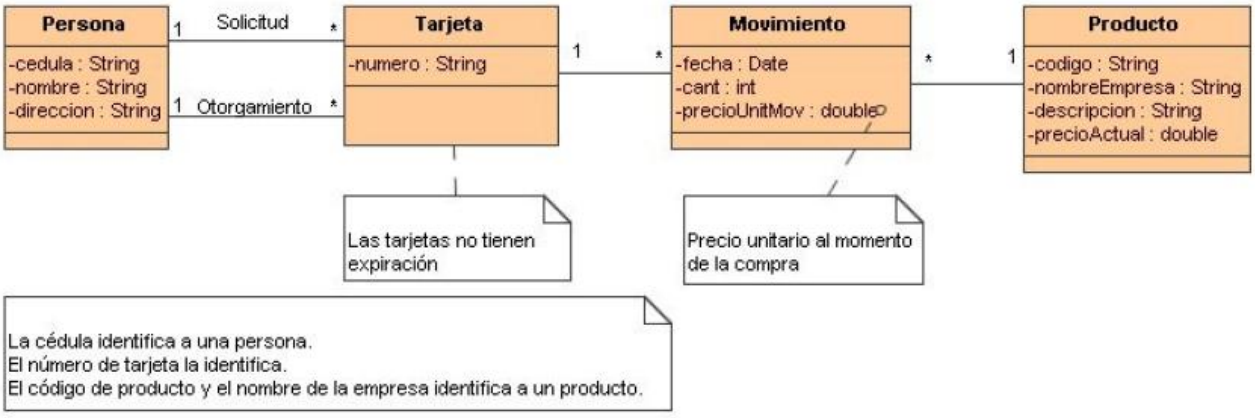
i. Explique la relación entre caso de uso, escenario y diagrama de secuencia del sistema.

ii. Explique la relación entre caso de uso y colaboración.

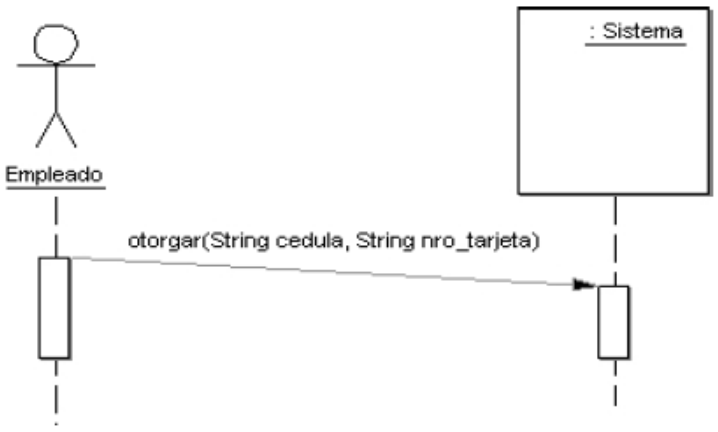
iii. Describa los 2 enfoques citados en el curso para diseñar una colaboración.

b) Se quiere terminar el análisis y realizar el diseño de una iteración de un sistema de gestión de solicitudes de tarjetas y registro de movimientos usando las mismas. A continuación se presentan el modelo conceptual y los casos de usos (junto con un diagrama de secuencia del sistema del curso típico de eventos) de esta iteración.

Modelo conceptual

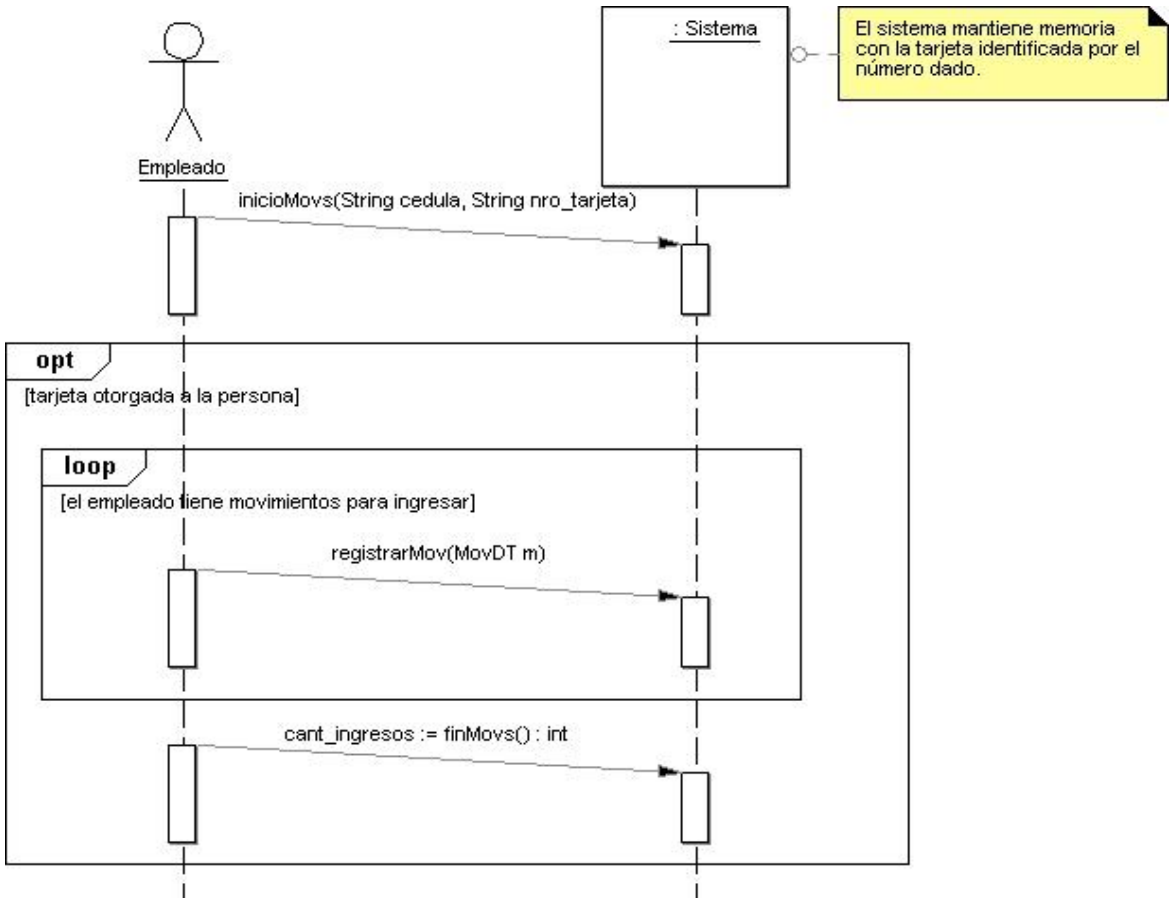


Nombre	Otorgamiento de tarjeta	**Actores**	Empleado
Sinopsis	El empleado indica al sistema el otorgamiento de una tarjeta conocida por su número a la persona identificada por su cédula de identidad. El sistema chequea la existencia de una solicitud de dicha tarjeta por la persona. En caso de que exista, se elimina la solicitud y se registra el otorgamiento. En caso contrario, la operación no debe tener efecto alguno sobre el sistema.		



void otorgar(cedula : String, nro_tarjeta : String)
pre: existe una instancia de Persona con cédula de identidad = cedula
pre: existe una instancia de Tarjeta identificada por el número = nro_tarjeta
pre: no existe un link de Otorgamiento entre la instancia de Persona con cédula de identidad = cedula y la instancia de Tarjeta identificada por el número = nro_tarjeta
post: si existe un link de Solicitud entre la instancia de Persona con cédula de identidad = cedula y la instancia de Tarjeta identificada por el número = nro_tarjeta, entonces se elimina dicho link y se crea un link de Otorgamiento entre el mismo par de instancias.

Nombre	Ingreso de movimientos	**Actores**	Empleado
Sinopsis	El caso de uso comienza cuando el empleado quiere ingresar un conjunto de movimientos de compras indicando la tarjeta y la cédula de identidad de la persona que hizo la compra para que el sistema realice el chequeo de que la tarjeta ha sido otorgada a esa persona. Luego, el empleado agrega uno a uno los movimientos. Durante este proceso, si el producto al cual se refiere el movimiento no existe, se agrega al sistema. Finalmente, el empleado le indica al sistema el fin del ingreso de movimientos. El sistema le devuelve la cantidad de movimientos ingresados.		



void inicioMovs(cedula : String, nro_tarjeta : String)
pre: existe una instancia de Persona con cédula de identidad = cedula
pre: existe una instancia de Tarjeta identificada por el número = nro_tarjeta
post: el sistema crea un link a la instancia de Tarjeta identificada por nro_tarjeta como la tarjeta actual
post: el sistema mantiene en memoria la cantidad ingresada y le asigna el valor 0

-- Notar que ni en el modelo conceptual ni en los casos de usos se destaca una relación entre el precioUnitMov y el precioActual del Producto.

void registrarMov(m : MovDT)
pre: el sistema tiene una instancia de Tarjeta como tarjeta actual
post: se crea una instancia de Movimiento con valor de atributo fecha = m.getFecha(), cant = m.getCantidad() y precioUnitMov = m.getPrecio()
post: se crea un link entre la tarjeta actual recordada por el sistema y la instancia de Movimiento creada
post: se crea un link entre la instancia de Movimiento creada y la instancia de Producto identificada por el código = m.getCodigoProducto() y nombre de empresa = m.getNombreEmpresa()
post: si y sólo si no existe un instancia de Producto identificada por el código = m.getCodigoProducto() y nombre de empresa = m.getNombreEmpresa(), se crea una instancia de Producto con valor de atributo codigo = m.getCodigoProducto(), nombreEmpresa = m.getNombreEmpresa(), descripcion = m.getDescripcion() y precioActual = m.getPrecio()
post: se incrementa en 1 la cantidad ingresada

int finMovs()
pre: el sistema tiene una instancia de Tarjeta como tarjeta actual
post: el sistema elimina el link a la instancia de Tarjeta identificada por nro_tarjeta como la tarjeta actual
post: devuelve como resultado el valor de cantidad ingresada

Se pide:

i. Realizar los diagramas de comunicación de las operaciones teniendo en cuenta los criterios GRASP. Mencionar explícitamente los criterios GRASP usados.

ii. Realice el DCD incluyendo las clases que considere necesarias.

Problema 3 (25 puntos) - Solución

a)

i. *Explique la relación entre caso de uso, escenario y diagrama de secuencia del sistema.*

Para un caso de uso pueden existir varios escenarios (tipicos y alternativos). Cada escenario de un caso de uso se puede representar con un DSS.

ii. *Explique la relación entre caso de uso y colaboración.*

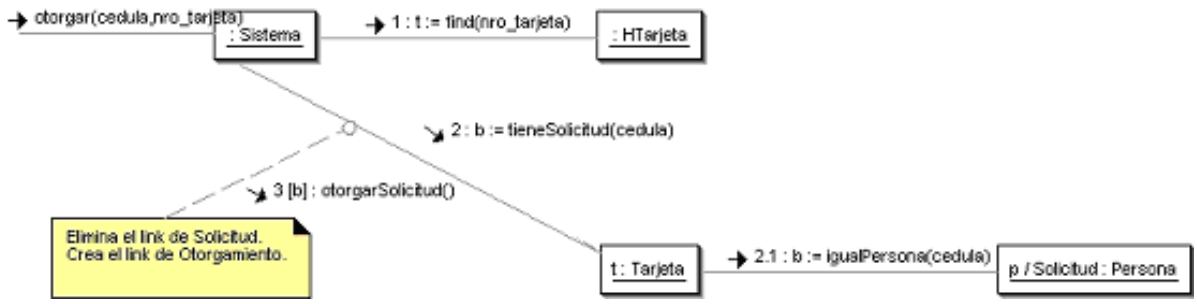
Una colaboración realiza uno o varios casos de uso.

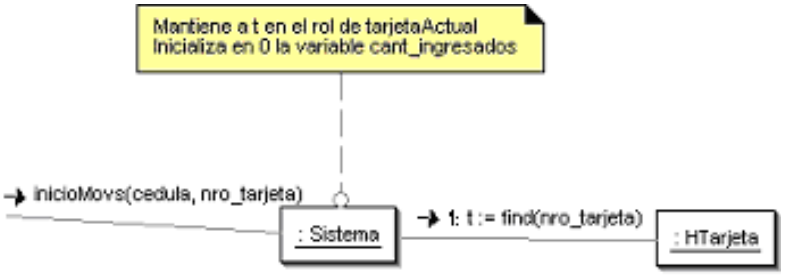
iii. *Describa los 2 enfoques citados en el curso para diseñar una colaboración.*

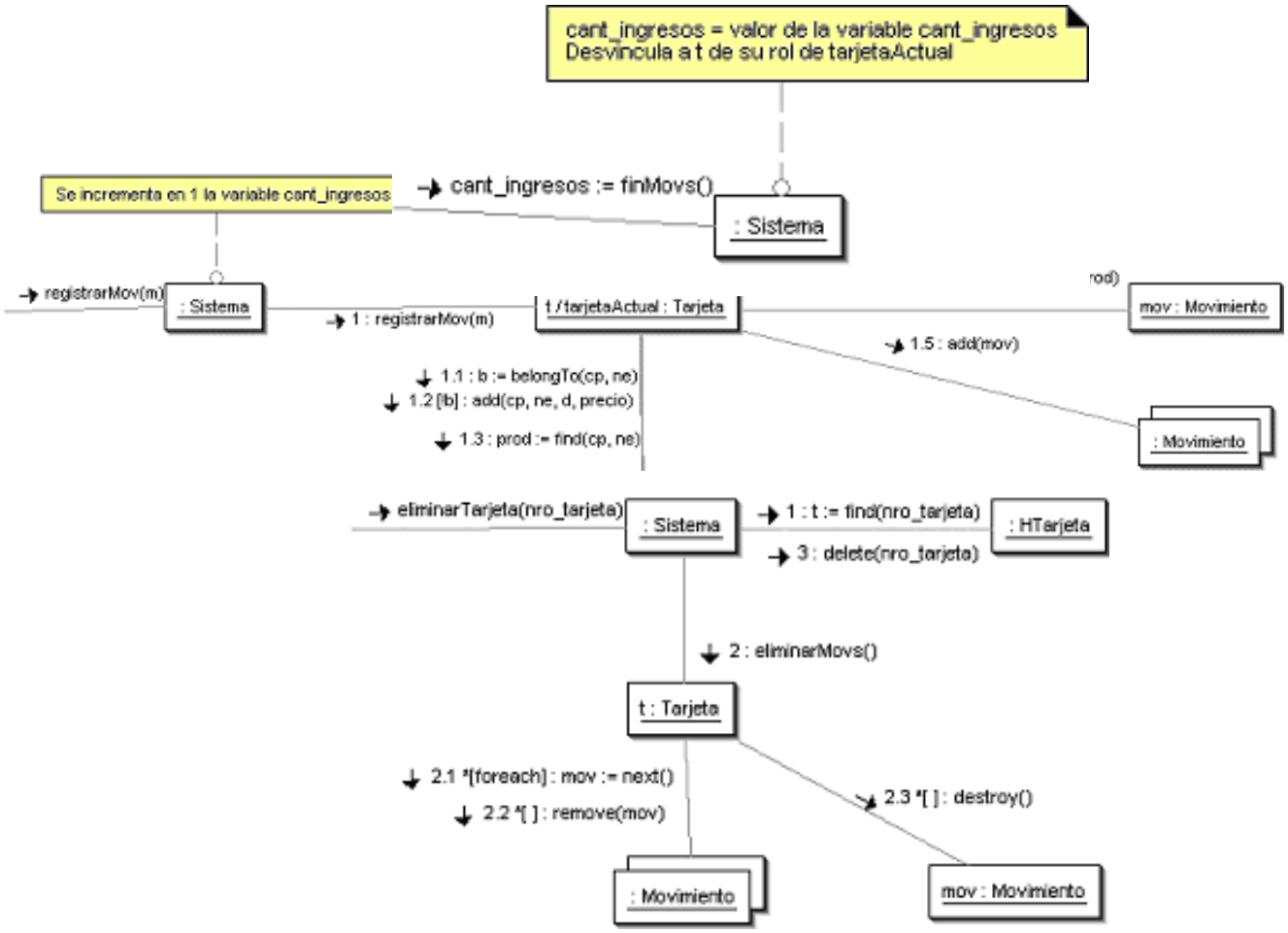
- Definir primero la estructura y luego generar las diferentes interacciones respetando dicha estructura.
- Definir las interacciones (teniendo en cuenta los criterios GRASP) y luego definir la estructura necesaria para que dichas interacciones puedan ocurrir.

b)

i. Diagramas de Comunicación



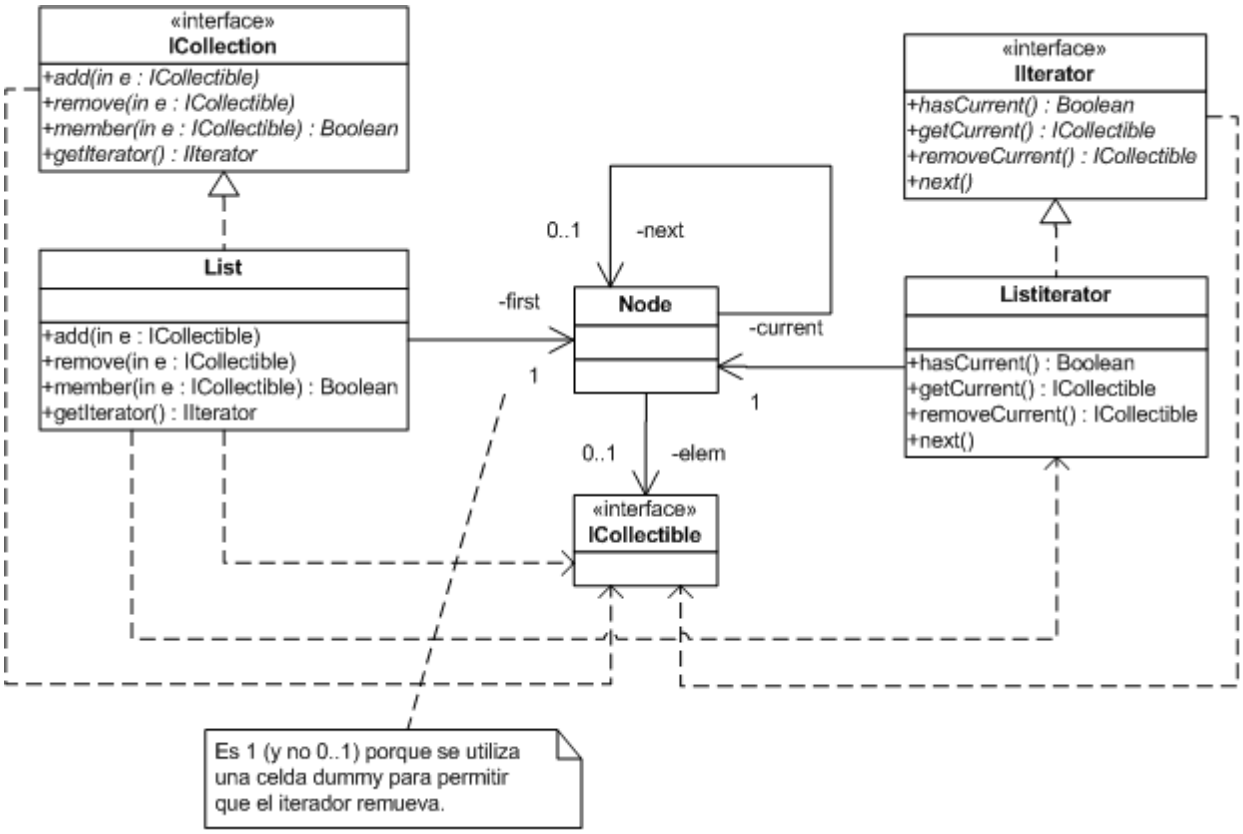

Mantiene a t en el rol de tarjetaActual
Inicializa en 0 la variable cant_ingresados
inicioMovs(cedula, nro_tarjeta)
: Sistema
1: t := find(nro_tarjeta)
: HTarjeta


cant_ingresos = valor de la variable cant_ingresos
Desvincula a t de su rol de tarjetaActual
Se incrementa en 1 la variable cant_ingresos
cant_ingresos := finMovs()
: Sistema
registrarMov(m)
: Sistema
1 : registrarMov(m)
t /tarjetaActual : Tarjeta
mov : Movimiento
1.5 : add(mov)
1.1 : b := belongTo(cp, ne)
1.2 [b] : add(cp, ne, d, precio)
1.3 : prod := find(cp, ne)
: Movimiento
eliminarTarjeta(nro_tarjeta)
: Sistema
1 : t := find(nro_tarjeta)
3 : delete(nro_tarjeta)
: HTarjeta
2 : eliminarMovs()
t : Tarjeta
2.1 *[foreach] : mov := next()
2.2 *[ ] : remove(mov)
2.3 *[ ] : destroy()
: Movimiento
mov : Movimiento

Problema 4 (25 puntos)

a) Explique brevemente que es una colección genérica y una colección concreta (o tipada).
Indique cómo se relacionan ambos conceptos a nivel de diseño.

b) Considere el siguiente DCD:



Se pide:

Implementar en C++ la interfaz ICollection y la clases List y Node.

OBSERVACION:
No es necesario incluir directivas al preprocesador en el código.