# Programación 4

## EXAMEN JULIO 2010

Por favor siga las siguientes indicaciones:

- Escriba con lápiz.
- Escriba las hojas de un solo lado.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Recuerde entregar su número de examen junto al examen.

### Problema 1 (35 puntos)

a) Se quiere desarrollar un software para registrar las actuaciones de los tenistas profesionales en diferentes torneos del circuito mundial, por lo que se ha relevado la siguiente realidad:

Para cada tenista interesa su nombre (lo identifica) y su nacionalidad. Los tenistas podrán jugar tenis de forma individual o en pareja. A cada pareja de tenistas se le otorgará un número el cual la identificará, independientemente de los torneos en los que participe. Se permite a un tenista formar varias parejas.

De cada torneo se desea conocer su nombre (lo identifica), el tipo de cancha en la que se juega (polvo de ladrillo, sintético o pasto), la cantidad máxima de tenistas que disputarán el torneo y el tipo de torneo que podrá ser Single o Doble.

Todo torneo se realiza una vez por año, y en cada edición del mismo interesará registrar, además del año de realización, los tenistas (o parejas) que participaron y su actuación en dicho torneo. La actuación será el número de puesto que obtuvo el tenista o la pareja en el torneo (sin repetirse el puesto). En las ediciones de torneos Single participan tenistas de forma individual, mientras que en los de tipo Doble participan parejas. Cabe señalar que no se permite participar a un tenista en dos o más parejas para la misma edición de un torneo de tipo Doble.

Se pide: modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML y expresar todas las restricciones del modelo en lenguaje natural.

b) Dado el siguiente contrato parcial

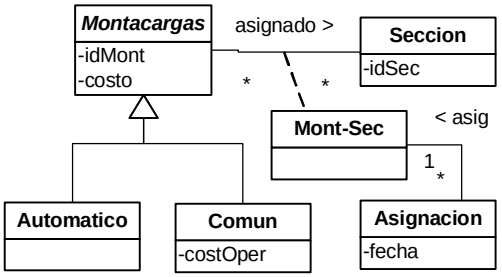
| | |
|---|---|
| Operación | `ingresarActuaciónTorneoSingle(nomTn:string,nomTr:string, año:int, nroPuesto:int)` |
| Parámetros | `nomTn: nombre del tenista`<br>`nomTr: nombre del torneo`<br>`año: año correspondiente a la edición del torneo de nombre nomTr`<br>`nroPuesto: número de puesto que corresponde a la actuación de tenista` |
| Descripción | `Permite ingresar la actuación de un tenista, mediante el número de puesto conseguido, a una edición existente de un torneo single.` |

Se pide: agregar las pre- y post-condiciones en lenguaje natural al contrato anterior, de acuerdo al modelo de dominio realizado en la parte a).

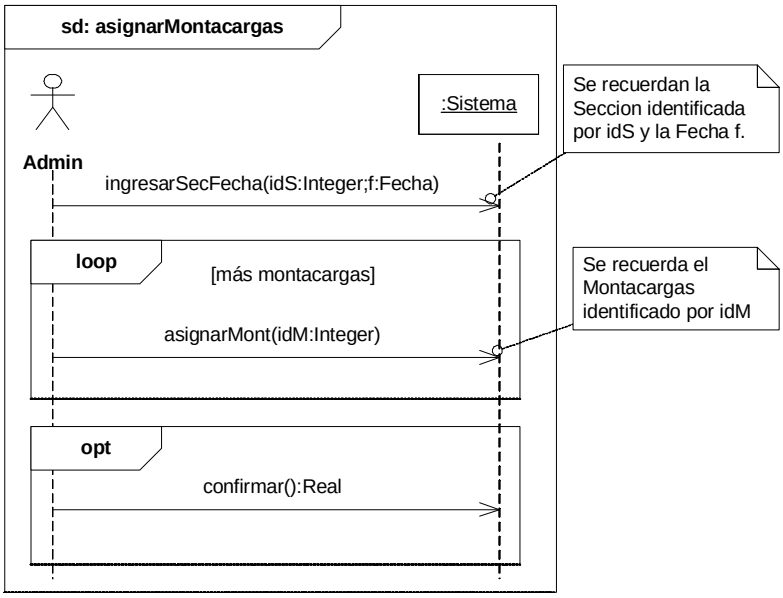
Problema 2 (30 puntos)

Se está modelando parte de un sistema de gestión de recursos para una empresa que brinda servicios de almacenamiento y distribución. La empresa cuenta con una flota de montacargas que pueden ser de tipo automático (no requieren operario) o común. Todos los montacargas tienen un costo diario de operación; los comunes además tienen un costo diario de operación por concepto del personal requerido para operarlos. Los montacargas se asignan a secciones de la empresa, interesando registrar la fecha de cada asignación de cada montacargas a cada sección.

A partir de esta realidad se ha realizado el siguiente diagrama de modelo de dominio con las restricciones en lenguaje natural. Además, se ha realizado el Diagrama de Secuencia del Sistema para el caso de uso de asignación de montacargas a secciones de la empresa.




- Los montacargas y las secciones se identifican por sus atributos idMont e idSec, respectivamente.
- Un montacargas no puede estar asignado a una misma sección más de una vez en la misma fecha.




Las operaciones del sistema del DSS anterior tienen las siguientes pre y post condiciones:

| Operación | ingresarSecFecha(idS : Integer, f : Fecha) |
|---|---|
| Pre y post condiciones | |
| pre: Existe una instancia de Seccion cuyo atributo idSec coincide con idS.<br><br>post: Se recuerdan en la memoria del sistema, la instancia de Seccion cuyo atributo idSec coincide con idS y el valor f. | |

| Operación | asignarMont(idM : Integer) |
|---|---|
| Pre y post condiciones | |
| pre: Existe una instancia de Montacargas cuyo atributo idMont coincide con idM.<br><br>pre: No existe en la memoria del sistema una instancia de Montacargas cuyo atributo idMont coincida con idM.<br><br>pre: El Montacargas identificado por idM no está asignado la Sección recordada en la fecha recordada.<br><br>post: Se recuerda en la memoria del sistema, la instancia de Montacargas cuyo atributo idMont coincide con idM. | |

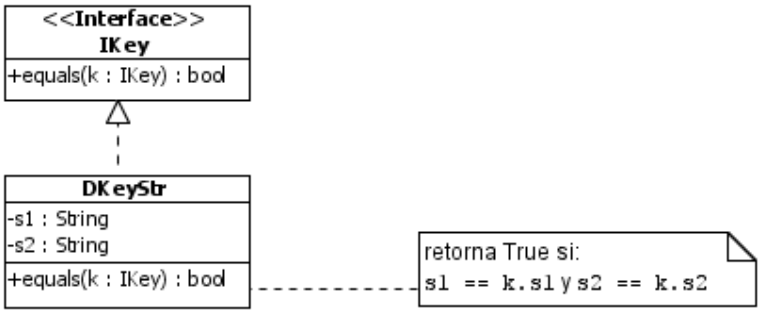
| Operación | confirmar() : Real |
|---|---|
| Pre y post condiciones | |
| pre: Existen en la memoria del sistema una instancia de Seccion denominada s, una fecha denominada f y una colección de instancias de Montacargas denominada ms.<br><br>post: Para cada instancia m de Montacargas en ms: Sea montSec la instancia del tipo asociativo Mont-Sec que asocia a m con s (si no existe se crea); se crea una nueva instancia de Asignacion asociada a montSec, cuyo atributo fecha coincide con f.<br><br>post: El resultado es la suma de los costos diarios de operación de todos los montacargas pertenecientes a la colección ms. El costo de un montacargas automático es el atributo costo mientras que el de uno común es la suma de los atributos costo y costOper. | |

Se pide:

a) Realizar los Diagramas de Comunicación de las tres operaciones descritas.
b) Realizar el Diagrama de Clases de Diseño resultante.

## Problema 3 (35 puntos)

a) Considere una clase llamada `DKeyStr` que almacena un par de strings s1 y s2 e implementa la interfaz `IKey` según el siguiente diseño.



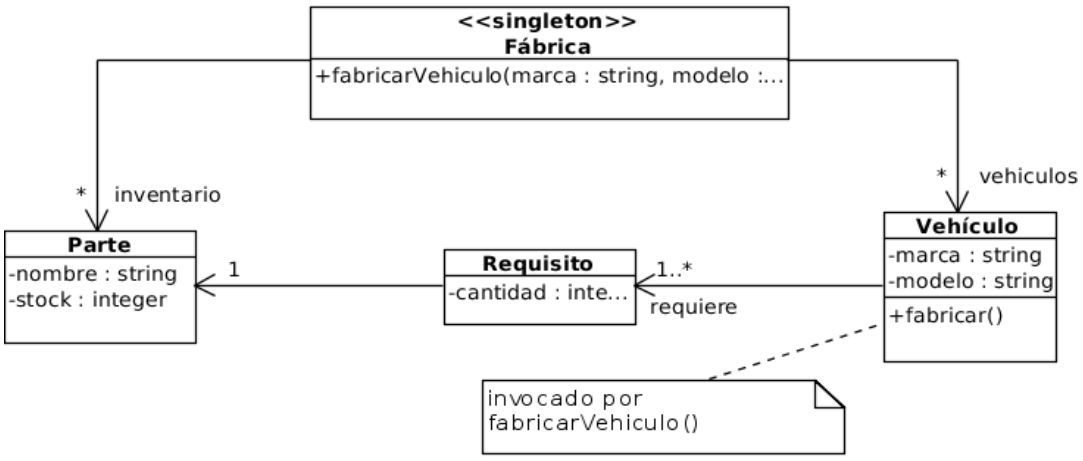

Se pide: Implemente en C++ los .h y .cpp de `DKeyStr`. Asuma que la clase `String` implementa el operador == que devuelve `True` si dos instancias contienen el mismo contenido.

b) Se le ha pedido implementar un sistema para una fábrica de automóviles. Los vehículos se identifican conjuntamente por su marca y su modelo (por ejemplo Fiat Uno). Para construir cada automóvil se necesita cierta cantidad de partes que se identifican por su nombre. De cada parte se conoce además su stock disponible. Para fabricar un vehículo tiene que haber stock suficiente de cada una de las partes requeridas, y cada vez que se construye un vehículo, el stock de las partes usadas decrece.

Como parte del análisis se generó el siguiente contrato de la clase Fábrica.

| Operación | `fabricarVehiculo(marca: String, modelo: String)` |
|---|---|
| Descripción | `Fabrica un vehículo decrementando el stock de partes usadas para fabricarlo, según los requisitos de partes para el vehículo.` |
| Pre y poscondiciones | |
| `pre: Existe una instancia de Vehículo v cuyo atributo marca es igual a marca y cuyo atributo modelo es igual a modelo.`<br><br>`pre: Para cada instancia de Requisito r asociado a v se cumple que el atributo cantidad de r no es mayor al valor del atributo stock de la instancia Parte que tiene asociada.`<br><br>`post: Para cada instancia de Parte asociada a las instancia de Requisito r anteriores, se decrementa el valor del atributo stock en N unidades, siendo N el valor del atributo cantidad de r.` | |

A partir de dicho contrato se generó el siguiente diseño que debe ser respetado en la implementación.




Se pide:

Implementar en C++:

i. El .h y .cpp de la clase `Vehículo`. No es necesario incluir en la implementación constructores, destructores ni operaciones get y set.
ii. El .h y el .cpp de la clase `Fabrica`.

Incluya explícitamente la verificación de las precondiciones en la implementación de la operación `fabricarVehiculo()`, lanzando excepciones en caso de que no se cumplan.

Observaciones:

- Asuma que existen implementaciones estándar de las interfaces `ICollection`, `IIterator`, `IDictionary` e `IKey`.
- Asuma la existencia de la interface `ICollectible`.
- Asuma la existencia de una clase `Exception` para manejar las excepciones de las operaciones.
- No es necesario incluir directivas de preprocesador.