Ejercicio 1 (35 puntos)

a) Defina brevemente qué es un contrato de software e indique 3 tipos de condiciones que se expresan en las postcondiciones de un contrato.

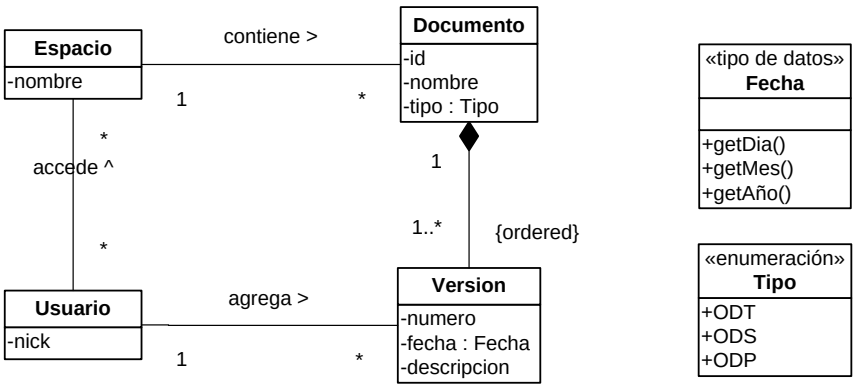
Un contrato de software especifica el comportamiento o efecto de una operación, determinando derechos y obligaciones para el proveedor y consumidor de la operación.
Tipos de condiciones que se expresan en un contrato:
- creación de objetos
- eliminación de objetos
- conexión de objetos
- desconexión de objetos
- modificación del valor de los atributos de objetos

b) Se desea implementar un software de gestión de documentos, para el cual se ha relevado lo siguiente:

El gestor de documentos está organizado lógicamente en espacios. Cada espacio se identifica por un nombre, contiene usuarios que podrán acceder al espacio, y podrá tener documentos que agreguen los usuarios con acceso al espacio. El usuario se identifica mediante un nick. Cada documento tiene número identificador, nombre, espacio al que pertenece y tipo de documento (permite solamente las extensiones ODT, ODS y ODP). El documento se compone de un conjunto de versiones. De la versión interesa el número de versión, la fecha, la descripción de la versión y el usuario que la agregó. Cabe señalar que cuando un usuario deja de tener acceso a un espacio, se desea mantener las versiones de los documentos que agregó a dicho espacio del gestor. Además, no se podrá permitir que un espacio tenga repetido un documento (con el mismo número).

Se pide:

i. Modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML.



ii. Expresar todas las restricciones del modelo en lenguaje natural.

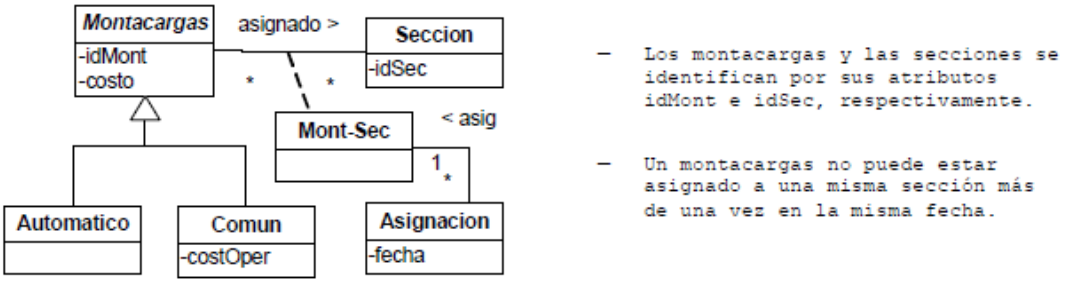
R1. No existen dos espacios con el mismo nombre.
R2. No existen dos usuarios con el mismo nick.
R3. No existen dos documentos con el mismo id.
R4. En el contexto de un documento, no existen dos versiones con el mismo número.
R5. Para un documento, los números de sus versiones son correlativos.
R6. La fecha de una versión de un documento es mayor o igual a las fechas de las versiones con numeración menor de dicho documento.

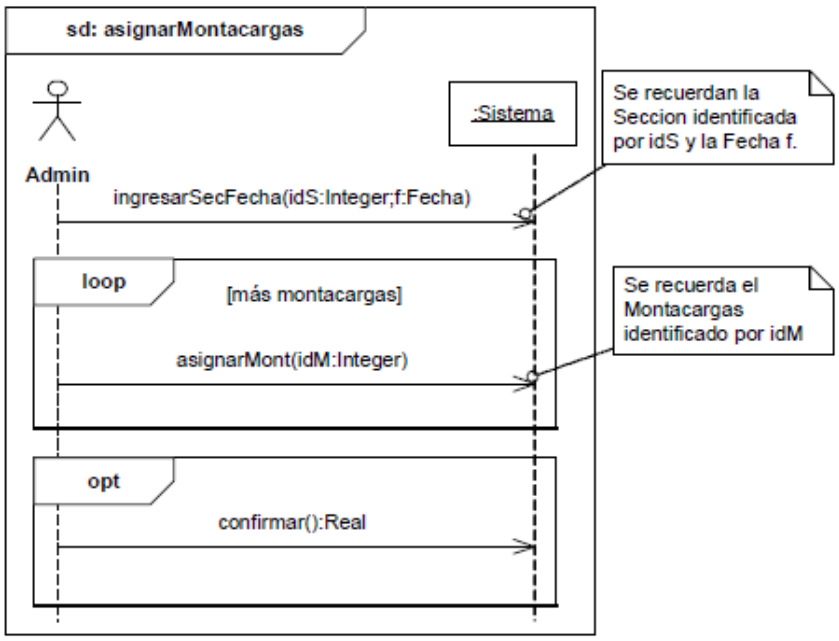
Ejercicio 2 (35 puntos)

a) Enumere y describa brevemente los criterios GRASP vistos en el curso.

b) 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.





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

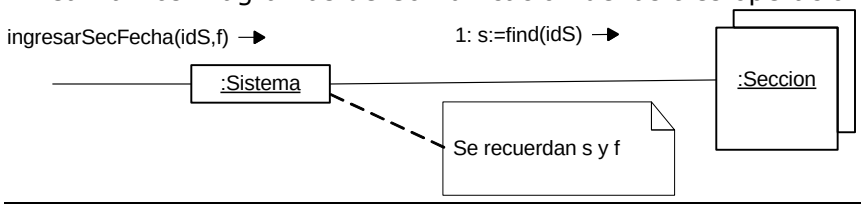
Operación	ingresarSecFecha(idS: Integer, f: Fecha)
Pre y Post Condiciones	
pre: Existe una instancia de Seccion cuyo atributo idSec coincide con idS. post: Se recuerdan en la memoria del sistema, la instancia de Seccion cuyo atributo idSec coincide con idS y el valor de f.	

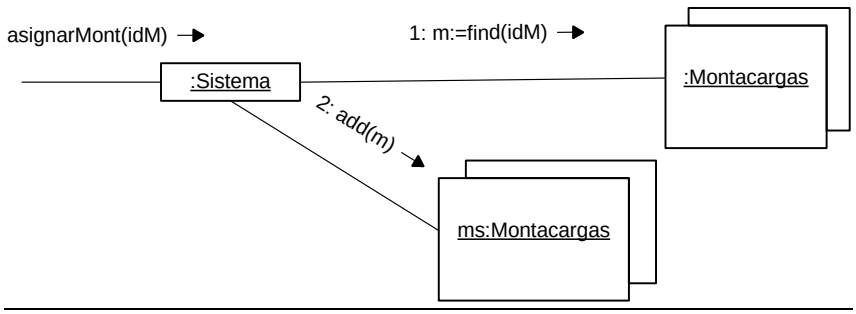
Operación	asignarMont(idM: Integer)
Pre y Post Condiciones	
pre: Existe una instancia de Montacargas cuyo atributo idMont coincide con idM. pre: No existe en la memoria del sistema una instancia de Montacargas cuyo atributo idMont coincida con idM. pre: El Montacargas identificado por idM no está asignado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

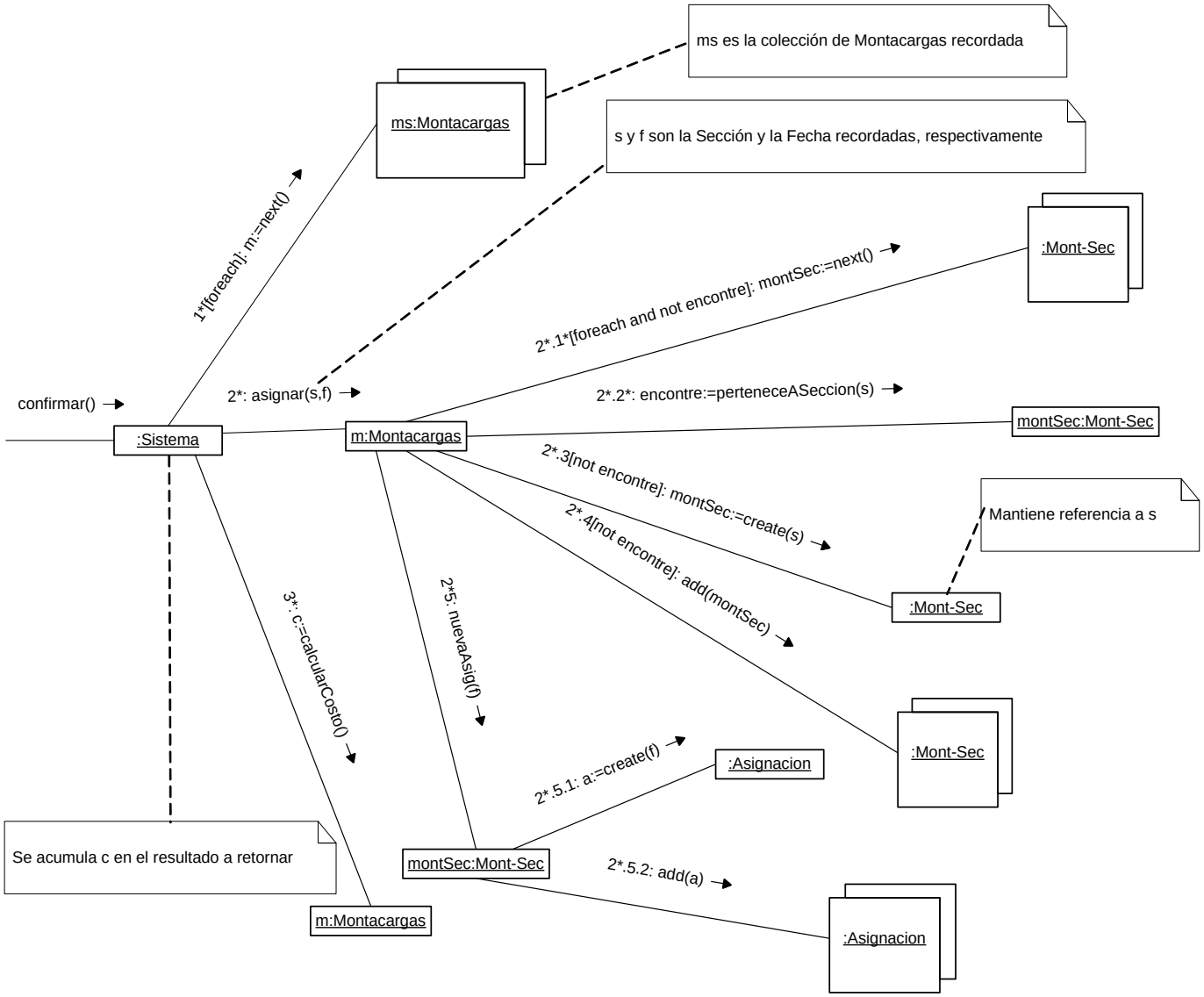
Operación	confirmar() : Real
Pre y Post Condiciones	
pre: Existen en la memoria del sistema una instancia de Seccion denominada s, una fecha denominada f y una colección de instancias de Montacargas denominada ms. post: Para cada instancia m de Montacargas en ms: Sea montSec la instancia del tipo asociativo 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. 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.	

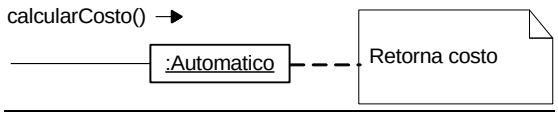
Se pide:

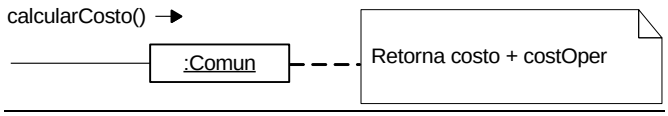
i. Realizar los Diagramas de Comunicación de las tres operaciones descritas.



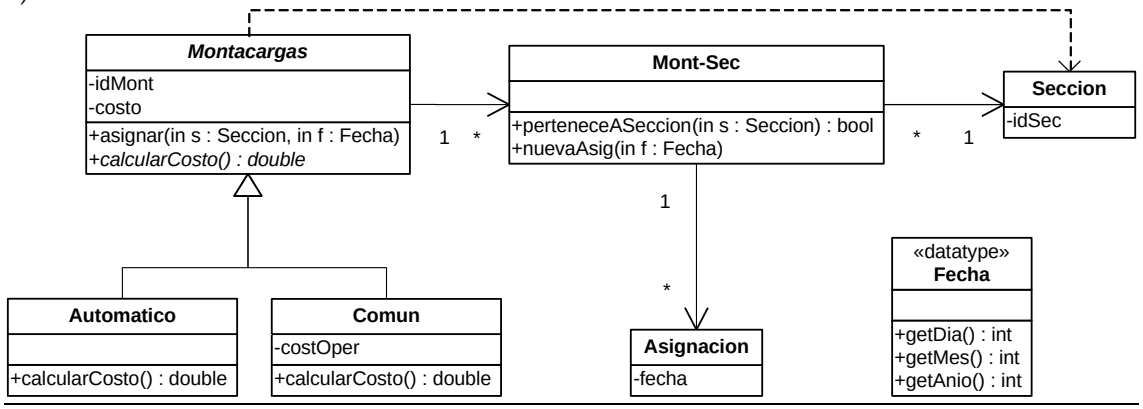



ms es la colección de Montacargas recordada
ms:Montacargas
s y f son la Sección y la Fecha recordadas, respectivamente
:Mont-Sec
1*[foreach]: m:=next()
2*.1*[foreach and not encontre]: montSec:=next()
confirmar()
2*: asignar(s,f)
2*.2*: encontre:=perteneceASeccion(s)
:Sistema
m:Montacargas
montSec:Mont-Sec
2*.3[not encontre]: montSec:=create(s)
Mantiene referencia a s
2*.4[not encontre]: add(montSec)
:Mont-Sec
3*: c:=calcularCosto()
2*5: nuevaAsig(f)
:Mont-Sec
2*.5.1: a:=create(f)
:Asignacion
Se acumula c en el resultado a retornar
montSec:Mont-Sec
2*.5.2: add(a)
m:Montacargas
:Asignacion


calcularCosto()
:Automatico
Retorna costo

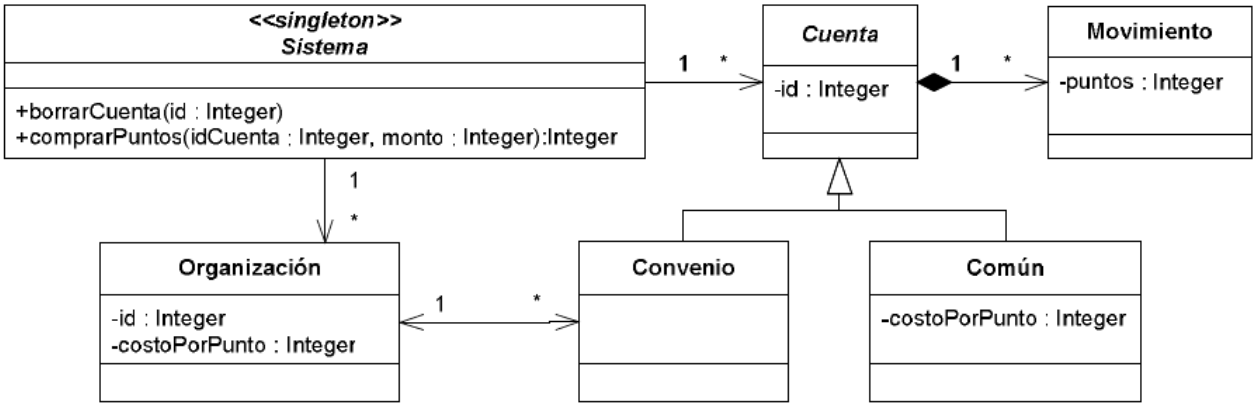

calcularCosto()
:Comun
Retorna costo + costOper

b)


Montacargas
-idMont
-costo
+asignar(in s : Seccion, in f : Fecha)
+calcularCosto() : double
1
*
Mont-Sec
+perteneceASeccion(in s : Seccion) : bool
+nuevaAsig(in f : Fecha)
*
1
Seccion
-idSec
1
*
Automatico
+calcularCosto() : double
Comun
-costOper
+calcularCosto() : double
Asignacion
-fecha
«datatype»
Fecha
+getDia() : int
+getMes() : int
+getAnio() : int

Ejercicio 3 (30 puntos)

Se le pidió que implemente un sistema para la gestión de las tarjetas de puntos de un supermercado. El sistema deberá registrar las cuentas de puntos de los clientes, los movimientos de puntos de cada cuenta (ganancia y uso de los puntos) y las organizaciones con la que el supermercado tiene convenios y promociones especiales. Se le ha entregado el siguiente diseño parcial que usted deberá respetar:



Se le ha entregado también la siguiente especificación:

Operación	Sistema::comprarPuntos(idCuenta : Integer, monto : Integer):Integer
Descripción	Crea una nueva instancia de Movimiento y la asocia a la uenta cuyo atributo id = idCuenta. El valor del atributo puntos se calcula como el monto dividido el costo por punto de la cuenta. Si es una cuenta Común, el costo por punto de la cuenta es el valor del atributo costoPorPunto. Si es una cuenta Convenio, es el valor del atributo con el mismo nombre en la instancia de Organización asociada. El valor retornado es el valor del atributo puntos de la nueva instancia. Si el monto es menor que el costo por punto, no se crea el movimiento y se lanza una excepción.

Se pide:

i) Implementar los .h de las clases Sistema, Cuenta, Común y Convenio.

```
class Sistema {

	public:
		static Sistema* getInstancia();
		~Sistema();
		void borrarCuenta(int id);
		int comprarPuntos(int idCuenta, int monto);
```

```
	private:
		Sistema();
		static Sistema* instancia;
		IDictionary* cuentas;
		IDictionary* organizaciones;
};

class Organizacion: public ICollectible {

	public:
		Organizacion(int id, int costoPorPunto);
		~Organizacion();
		void quitarConvenio(Convenio* convenio);
		int getCostoPorPunto();

	private:
		int id;
		int costoPorPunto;
		IDictionary* convenios;
};

class Cuenta: public ICollectible {

	public:
		Cuenta(int id);
		~Cuenta();
		int comprarPuntos(int monto);
		virtual int getCostoPorPunto() = 0;

	private:
		int id;
		ICollection* movimientos;
};

class Convenio: public Cuenta {

	public:
		Convenio(int id, Organizacion* org);
		~Convenio();
		int getCostoPorPunto();

	private:
		Organizacion* organizacion;
};




class Comun: public Cuenta {

	public:
		Comun(int id, int costo);
		int getCostoPorPunto();

	private:
		int costoPorPunto;
};
```

ii) Implementar en C++ la operación especificada anteriormente y todas aquellas operaciones necesarias para su implementación que pertenezcan a las clases Sistema, Cuenta, Común y Convenio.

Operación comprarPuntos.

```
int Sistema::comprarPuntos(int idCuenta, int monto) {
	KeyInteger* key = new KeyInteger(idCuenta);
	Cuenta* cuenta = (Cuenta*) cuentas->find(key);
	delete key;
	if (cuenta != NULL) {
		return cuenta->comprarPuntos(monto);
	} else {
		throw invalid_argument("No existe la cuenta");
	}
}

int Cuenta::comprarPuntos(int monto) {
	int costoPorPunto = getCostoPorPunto();
	if (monto < costoPorPunto)
		throw out_of_range("monto insuficiente");
	int puntos = monto / costoPorPunto;
	Movimiento* mov = new Movimiento(puntos);
	movimientos->add(mov);
	return puntos;
}

int Convenio::getCostoPorPunto() {
	return organizacion->getCostoPorPunto();
}

int Comun::getCostoPorPunto() {
	return costoPorPunto;
}

Movimiento::Movimiento(int puntos) : puntos(puntos) {
}
```