Programación 4

PARCIAL FINAL EDICIÓN 2007

SOLUCIÓN

Problema 1 (25 puntos)

a) Responda brevemente las siguientes preguntas

i) ¿Qué elementos están presentes en los diagramas de modelo de dominio y no en los diagramas de clases de diseño?

En el modelo de dominio están presentes las clases de asociación

ii) ¿Qué elementos están presentes en los diagramas de clases de diseño y no en los diagramas de modelo de dominio?

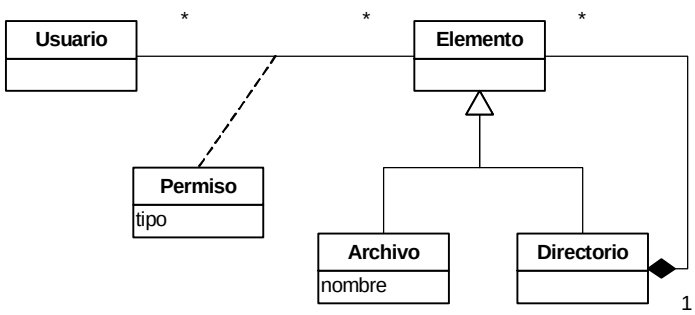
En el diagrama de clases de diseño aparecen: dependencias, navegabilidades, realizaciones, interfaces y operaciones

iii) ¿Qué técnicas existen para la identificación de conceptos del Modelo de Dominio?

Se pueden utilizar listas de categorías de conceptos e identificación de sustantivos.

b) Construir Modelos de Dominio (separados) para las siguientes realidades y presentarlos en diagramas utilizando UML. Las restricciones deben ser expresadas en lenguaje natural y en OCL.

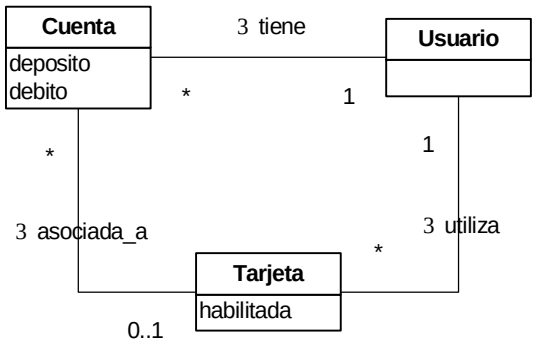
i)




```
-- Si un usuario tiene permiso sobre un directorio entonces
-- tiene permisos sobre los elementos de ese directorio
context Usuario inv:
self.elemento->forAll(e:Elemento | e.oclIsTypeOf(Directorio)
    implies self.elemento
       ->includesAll(e.oclAsType(Directorio).elemento))

-- El nombre identifica al archivo dentro de un directorio
context Directorio inv:
  self.elemento->forAll(e1,e2:Elemento |
           (e1.oclIsTypeOf(Archivo) and e2.oclIsTypeOf(Archivo)
           and e1 <> e2) implies e1.oclAsType(Archivo).nombre
                          <> e2.oclAsType(Archivo).nombre)
```

ii)




```
-- Un usuario utiliza tarjetas asociadas a cuentas que son de él
context Usuario inv:
  self.cuenta.tarjeta->notEmpty() implies
       self.cuenta.tarjeta->includesAll(self.tarjeta)

-- Si el balance es negativo la tarjeta debe estar deshabilitada
context Cuenta inv:
  (self.deposito - self.debito < 0) implies
              (self.tarjeta->notEmpty() implies
                          self.tarjeta.habilitada = false)
```

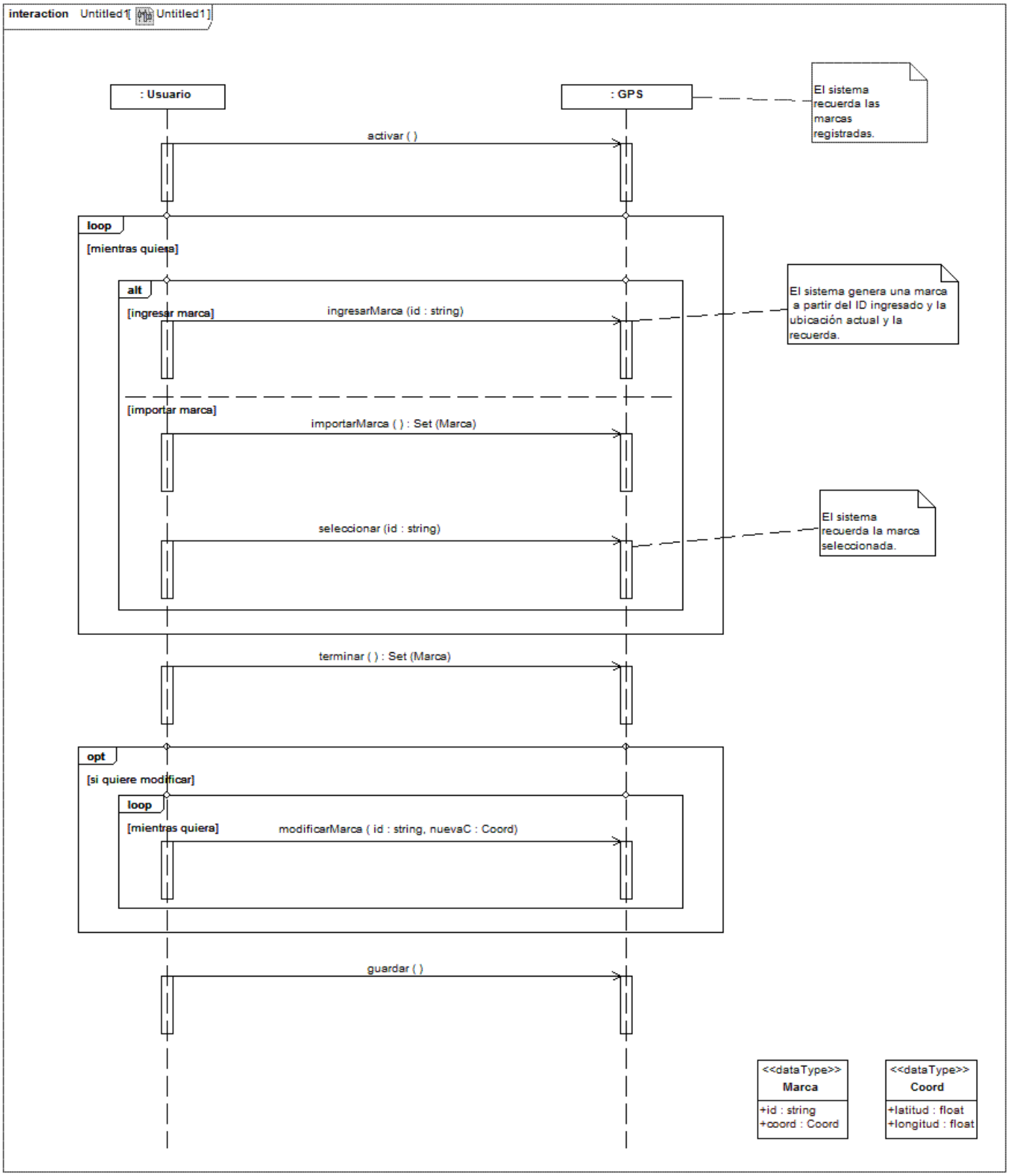
Problema 2 (25 puntos)

a) Conteste brevemente qué se refiere con "memoria del sistema".

Son los datos que el Sistema debe guardar temporalmente mientras se esté ejecutando un Caso de Uso. Representa el estado de la conversación entre usuario y Sistema.

b) .

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


interaction Untitled1 Untitled1
: Usuario
: GPS
El sistema recuerda las marcas registradas.
activar ( )
loop
[mientras quiera]
alt
[ingresar marca]
ingresarMarca (id : string)
El sistema genera una marca a partir del ID ingresado y la ubicación actual y la recuerda.
[importar marca]
importarMarca ( ) : Set (Marca)
seleccionar (id : string)
El sistema recuerda la marca seleccionada.
terminar ( ) : Set (Marca)
opt
[si quiere modificar]
loop
[mientras quiera]
modificarMarca (id : string, nuevaC : Coord)
guardar ( )
<<dataType>>
Marca
+id : string
+coord : Coord
<<dataType>>
Coord
+latitud : float
+longitud : float

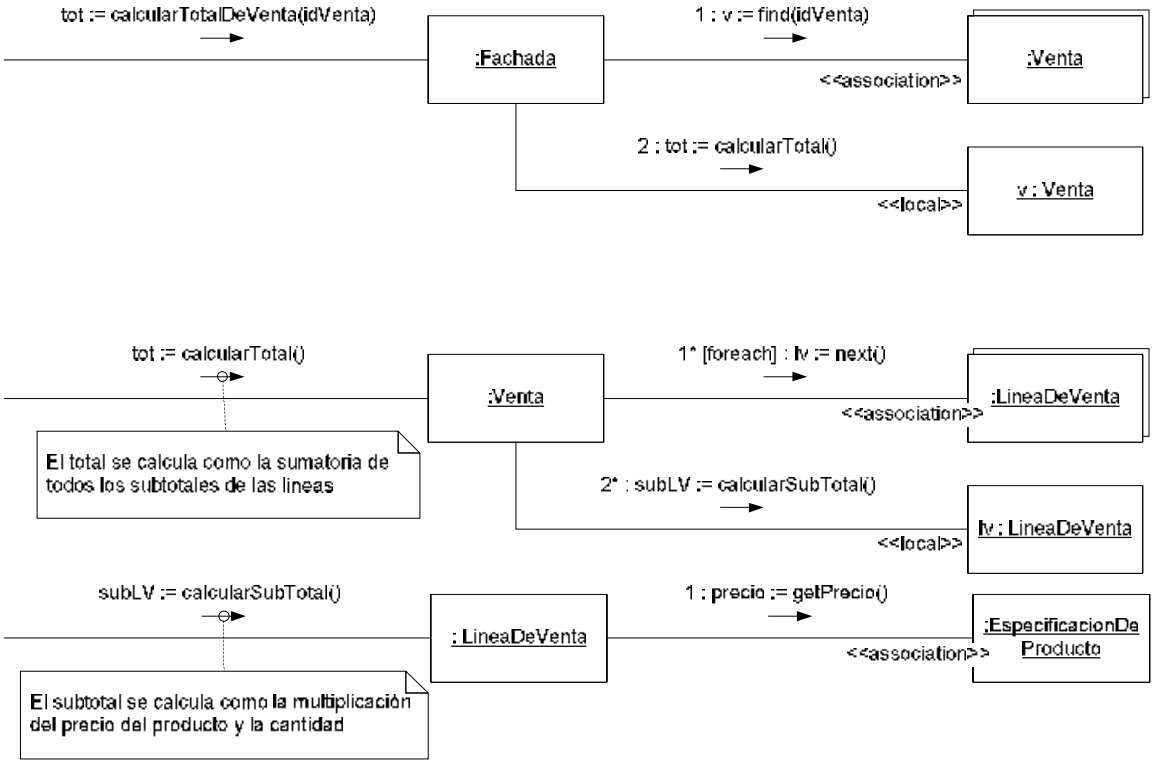
Problema 3 (25 puntos)

a) Describa brevemente las responsabilidades que pueden ser asignadas a una colección de objetos en un diagrama de comunicación.

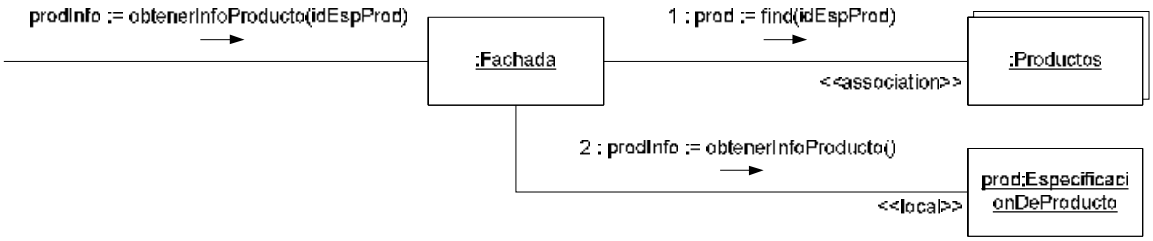
Las colecciones serán tratadas como meros contenedores de objetos por lo que no tendrán otra responsabilidad más que esa. Proveerán solamente operaciones que permitan administrar los objetos contenidos. En general las interfaces de Diccionario (add, remove, find, member, etc.) e Iterador (next, etc.) son suficientes para las colecciones.

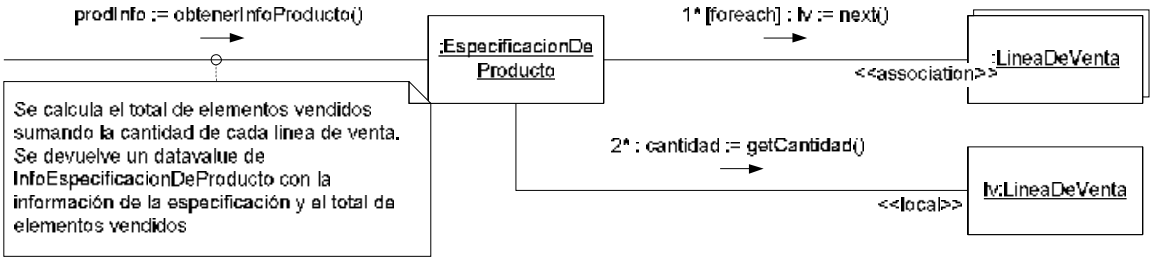
b) Realice los diagramas de comunicación para las operaciones que se especifican en los diagramas de secuencia. Los diagramas deben mostrar las visibilidades utilizadas, cumplir con los contratos y con las decisiones de diseño que ya han sido tomadas

i) Calcular total de venta.

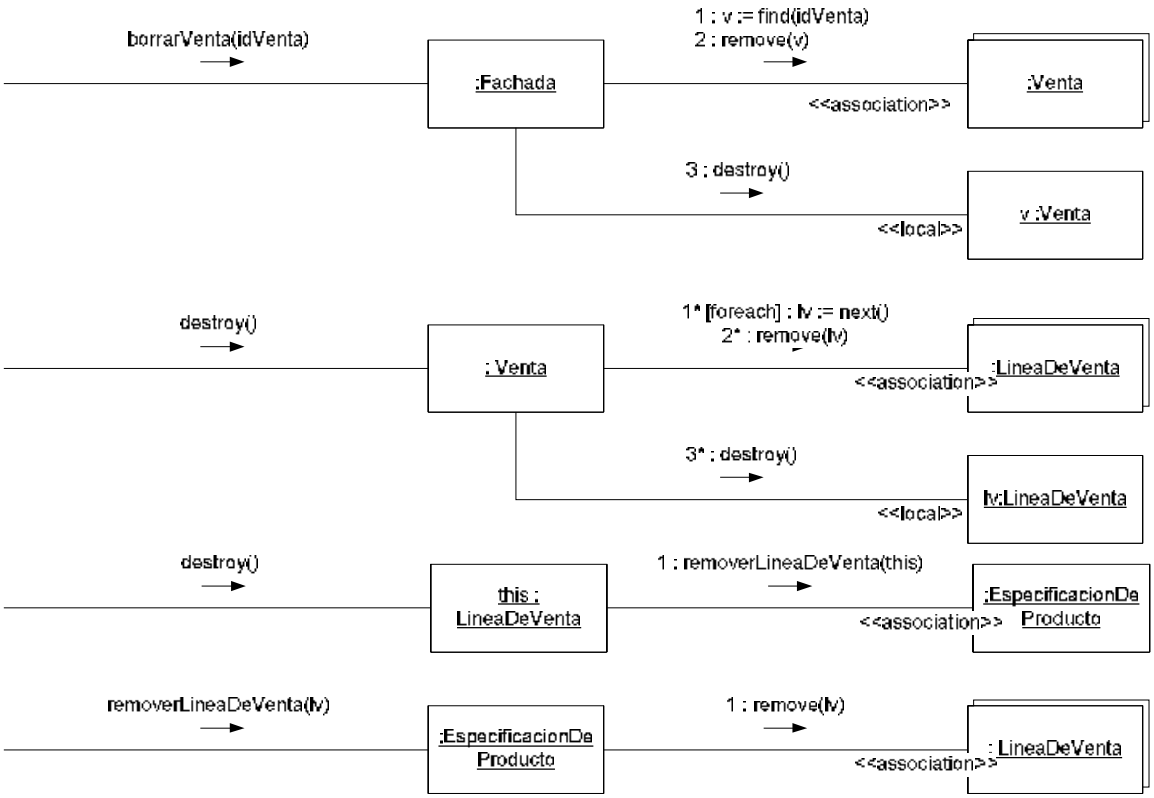


ii) Obtener Información de un Producto





iii) Borrar una venta



Problema 4 (25 puntos)

Implementar en C++ el diseño parcial dado, agregando lo que sea necesario para tener una solución completa.

Object.hh

```
class Object
{
public:
   virtual Object* copiar() = 0;
   virtual ~Object();
};
```

Object.cc

```
Object::~Object()
{
}
```

Celda.hh

```
class Celda : public Object
{
private:
   int fila;
   int columna;
   bool protegida;
public:
   Celda(int, int, bool);
   Object* copiar();
   virtual Object* copiaPrivada(int, int, bool);
};
```

Celda.cc

```
Celda::Celda(int f, int c, bool p) : fila(f), columna(c), protegida(p)
{
}

Object* Celda::copiar()
{
   // Si es protegida se copia sólo la parte de Celda
   if (this->protegida)
      return this->Celda::copiaPrivada(this->fila, this->columna, true);

   // Si no es protegida se copia según el tipo dinámico
   return this->copiaPrivada(this->fila, this->columna, false);
}

Object* Celda::copiaPrivada(int f, int c, bool p)
{
   return new Celda(f, c, p);
}
```

CeldaImagen.hh

```
class CeldaImagen : public Celda
{
private:
   Imagen* imagen;
public:
   CeldaImagen(Imagen*, int, int, bool);
   Object* copiaPrivada(int, int, bool);
   ~CeldaImagen();
};
```

CeldaImagen.cc

```
CeldaImagen::CeldaImagen(Imagen* i, int f, int c, bool p) : Celda(f, c, p),
imagen(i)
{
}

Object* CeldaImagen::copiaPrivada(int f, int c, bool p)
{
   return new CeldaImagen(Utils::copiarImagen(this->imagen), f, c, p);
}

CeldaImagen::~CeldaImagen()
{
   delete this->imagen;
}
```

Hoja.hh

```
class Hoja : public Object
{
private:
   ICollection* celdas;
   String nombre;
public:
   Hoja(String);
   Object* copiar();
   ~Hoja();
};
```

Hoja.cc

```
Hoja::Hoja(String n) : nombre(n), celdas(new List())
{
}

Object* Hoja::copiar()
{
   IIterator* i = this->celdas->getIterator();
   Hoja* hoja = new Hoja(this->nombre);

   // Se copian las celdas
   while (i->hasCurrent())
   {
      hoja->celdas->add(i->current()->copiar());
      i->next();
   }

   delete i;

   return hoja;
}

Hoja::~Hoja()
{
   IIterator* i = this->celdas->getIterator();

   // Se eliminan todas las celdas
   while (i->hasCurrent())
      delete i->removeCurrent();

   // Se elimina la colección
   delete this->celdas;

   delete i;
}
```

PortaPapeles.hh

```
class PortaPapeles
{
private:
   static PortaPapeles* instance;
   Object* contenido;
   PortaPapeles();
public:
   static PortaPapeles* getInstance();
   Object* getObjet();
   void setObjet(Object*);
};
```

PortaPapeles.cc

```
PortaPapeles* PortaPapeles::instance = NULL;

PortaPapeles::PortaPapeles()
{
   this->contenido = NULL;
}

PortaPapeles* PortaPapeles::getInstance()
{
   if (PortaPapeles::instance == NULL)
      PortaPapeles::instance = new PortaPapeles();

   return PortaPapeles::instance;
}

Object* PortaPapeles::getObjeto()
{
   if (this->contenido != NULL)
      return this->contenido->copiar();

   return NULL;
}

void PortaPapeles::setObjeto(Object* o)
{
   if (this->contenido != NULL)
      delete this->contenido;

   this->contenido = o;
}
```