Segundo Parcial 2010
Solución

Presentar la resolución del parcial:
· 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.
· · **Comience cada ejercicio en una hoja nueva.**
· El parcial es individual y sin material. **APAGUE SU CELULAR.**
· **Escriba con lápiz y de forma prolija.**
· Duración: 3 horas.

Problema 1 (10 puntos)

Pregunta 1)
Nombrar tres criterios GRASP y explicar brevemente en qué consiste cada uno.

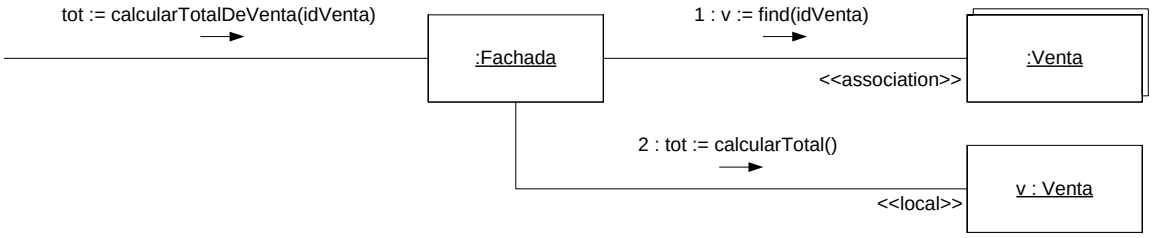
Pregunta 2)
Describa brevemente las responsabilidades que pueden ser asignadas a una colección de objetos en un diagrama de comunicación.

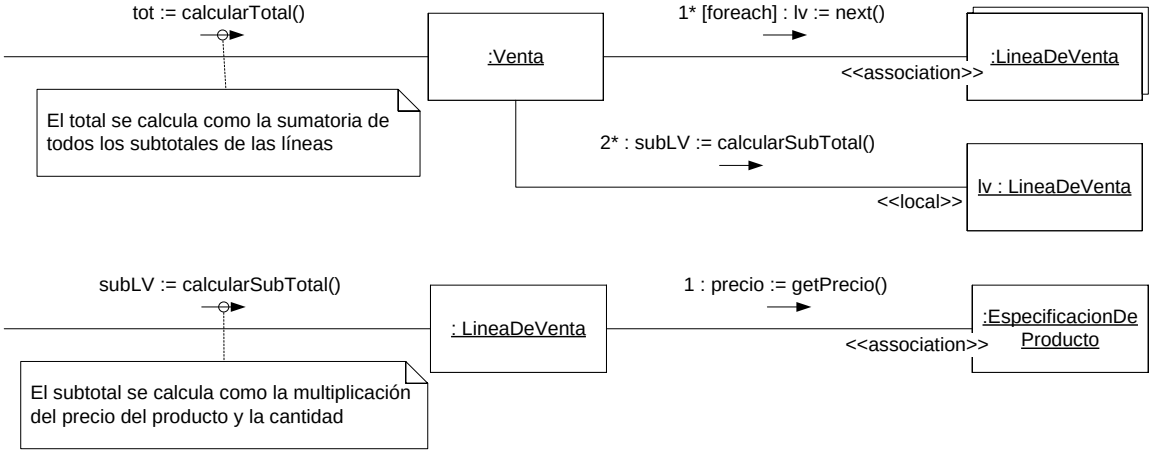
Pregunta 3)
Defina brevemente cómo se mapean a C++ las relaciones de generalización, realización y dependencia.

Problema 2 (25 puntos)

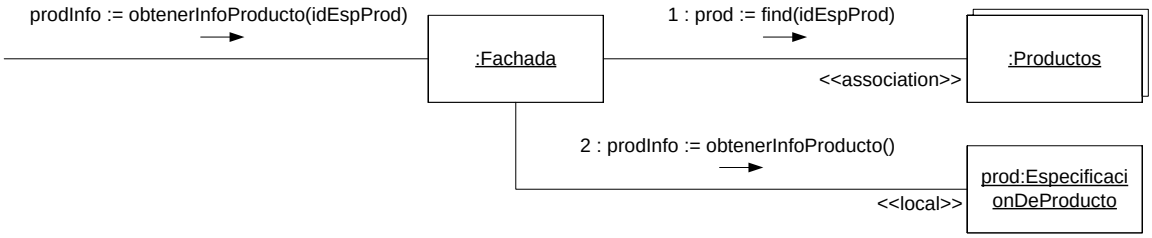
SE PIDE: Realice los diagramas de comunicación para las operaciones que se especifican en los diagramas de secuencia. Los diagramas deben mostrar las visibilidades Utilizadas, cumplir con los contratos y con las decisiones de diseño que ya han sido tomadas.

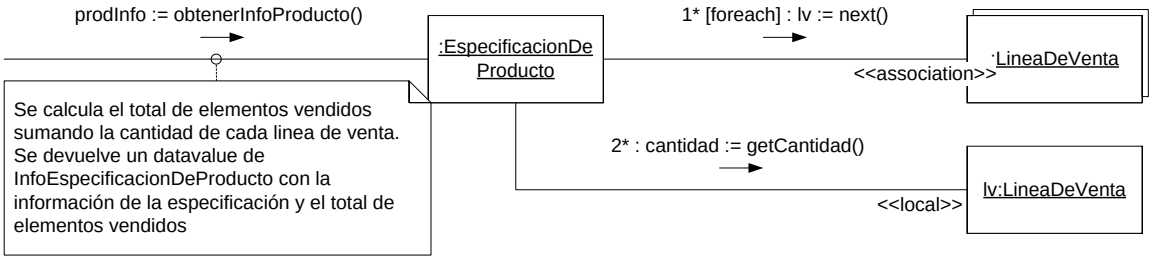
i) Calcular total de venta.



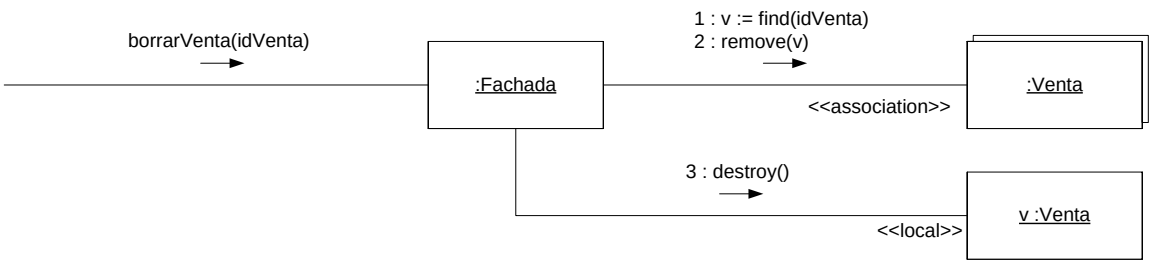


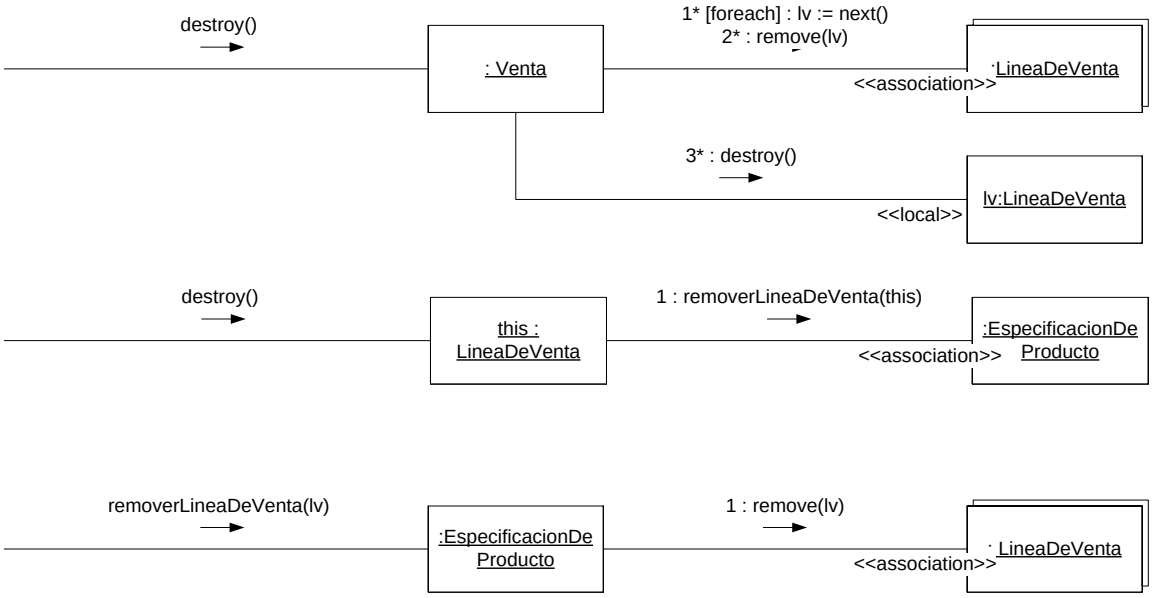
ii) Obtener Información de un Producto





iii) Borrar una venta





Problema 3 (25 puntos)

SE PIDE: Implementar en C++ completamente la colaboración salvo la clase `Cambios`. Del datatype `DataDocumento` es necesario incluir únicamente su módulo de definición, es decir su .h.

```
// DataDocumento.hh

class DataDocumento : public ICollectible
{
private:
      String nombre;
      String descripcion;
      int cantidadVersiones;

public:
      DataDocumento(String nom, String desc, int cantVersiones);
      void setNombre(String nom);
      void setDescripcion(String desc);
      void setCantidadVersiones(int cantVersiones);
      String getNombre();
      String getDescripcion();
      int getCantidadVersiones();
};
```

```
// Documento.hh
class Documento : public ICollectible
{
private:
      String nombre;
      String descripcion;

public:
      virtual DataDocumento getDataDocumento();
      virtual ~Documento();
};


// Documento.cc
DataDocumento Documento::getDataDocumento()
{
      return DataDocumento(this->nombre, this->descripcion, 1);
}

Documento::~Documento(){}


// Binario.hh
class Binario : public Documento
{
};


// TextoPlano.hh
class TextoPlano : public Documento
{
private:
      ICollection * cambios;

public:
      DataDocumento getDataDocumento();
      ~TextoPlano();
};


// TextoPlano.cc
DataDocumento TextoPlano::getDataDocumento()
{
      DataDocumento aux = Documento::getDataDocumento();
      aux.setCantidadVersiones(this->cambios->size() + 1);

      return aux;
}
```

```
TextoPlano::~TextoPlano()
{
	IIterator * i = this->cambios->getIterator();

	while (i->hasCurrent())
	{
		delete i->removeCurrent();
	}

	delete i;
	delete this->cambios;
}


// ControladorDocumentos.hh
class ControladorDocumentos
{
private:
	static ControladorDocumentos * instance;
	IStringDictionary * documentos;

	ControladorDocumentos();

public:
	static ControladorDocumentos * getInstance();

	ICollection * listarDocumentos();
	void borrarDocumento(String nombre);
};


// ControladorDocumentos.cc
ControladorDocumentos * ControladorDocumentos::instance = NULL;

ControladorDocumentos::ControladorDocumentos()
{
	this->documentos = new List();
}
```

```
ControladorDocumentos * ControladorDocumentos::getInstance()
{
	if (ControladorDocumentos::instance == NULL)
	{
		ControladorDocumentos::instance =
			new ControladorDocumentos();
	}

	return ControladorDocumentos::instance;
}

ICollection * ControladorDocumentos::listarDocumentos()
{
	IIterator * i = this->documentos->getIterator();
	List * resultado = new List();

	while (i->hasCurrent())
	{
		// Se crea una copia para devolver
		resultado->add(new DataDocumento(
			((Documento*)i->getCurrent())-
		>getDataDocumento()
		));

		i->next();
	}

	delete i;

	return resultado;
}

void ControladorDocumentos::borrarDocumento(String nombre)
{
	delete this->documentos->remove(nombre);
}
```