Programación 4

EXAMEN DICIEMBRE 2016 - SOLUCIÓN

Problema 1

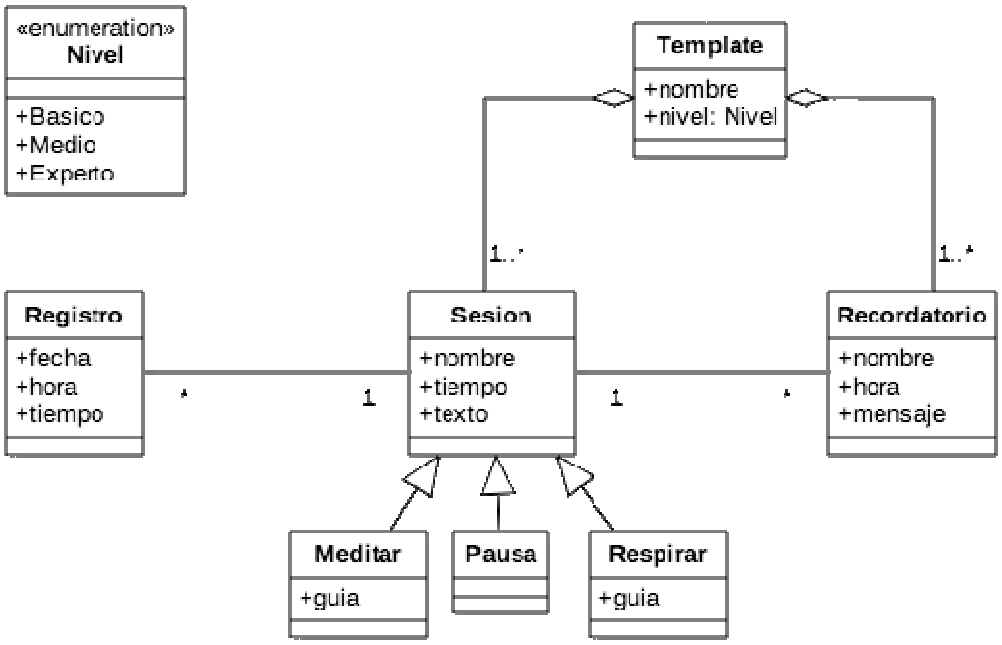
Parte a)

i. Modelado del Dominio y Definición del Comportamiento del Sistema.

ii. Es declarativa, ya que las PRE y POST condiciones no dicen el COMO sino el QUE.

Parte b)

i.



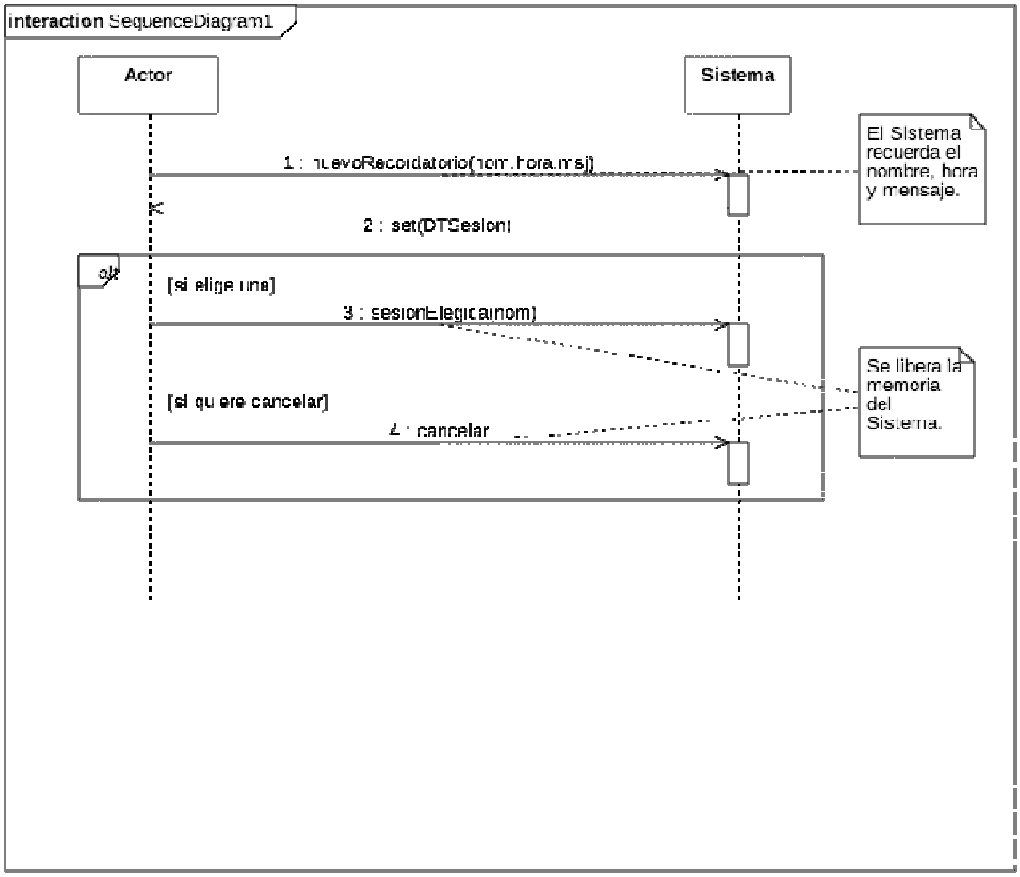
Obs.: debido al software de modelado utilizado, se le debe agregar una multiplicidad de * en ambas composiciones de Template.

Restricciones:

- El nombre identifica a la Sesion
- El nombre identifica al Recordatorio
- El nombre identifica al Template
- Los recordatorios de los templates deben ser sobre sesiones que también pertenezcan a ese template.

ii.

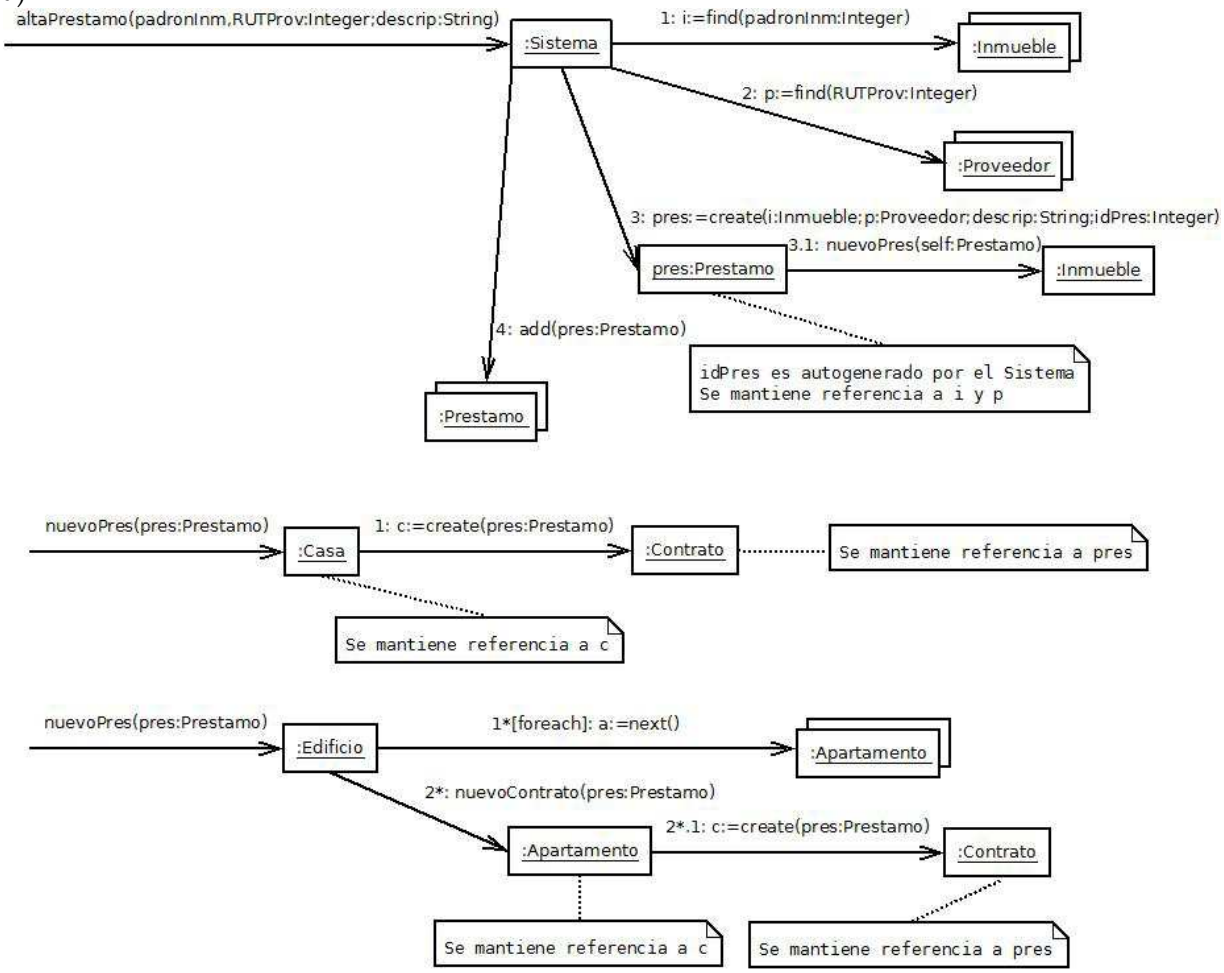
DSS Con memoria.



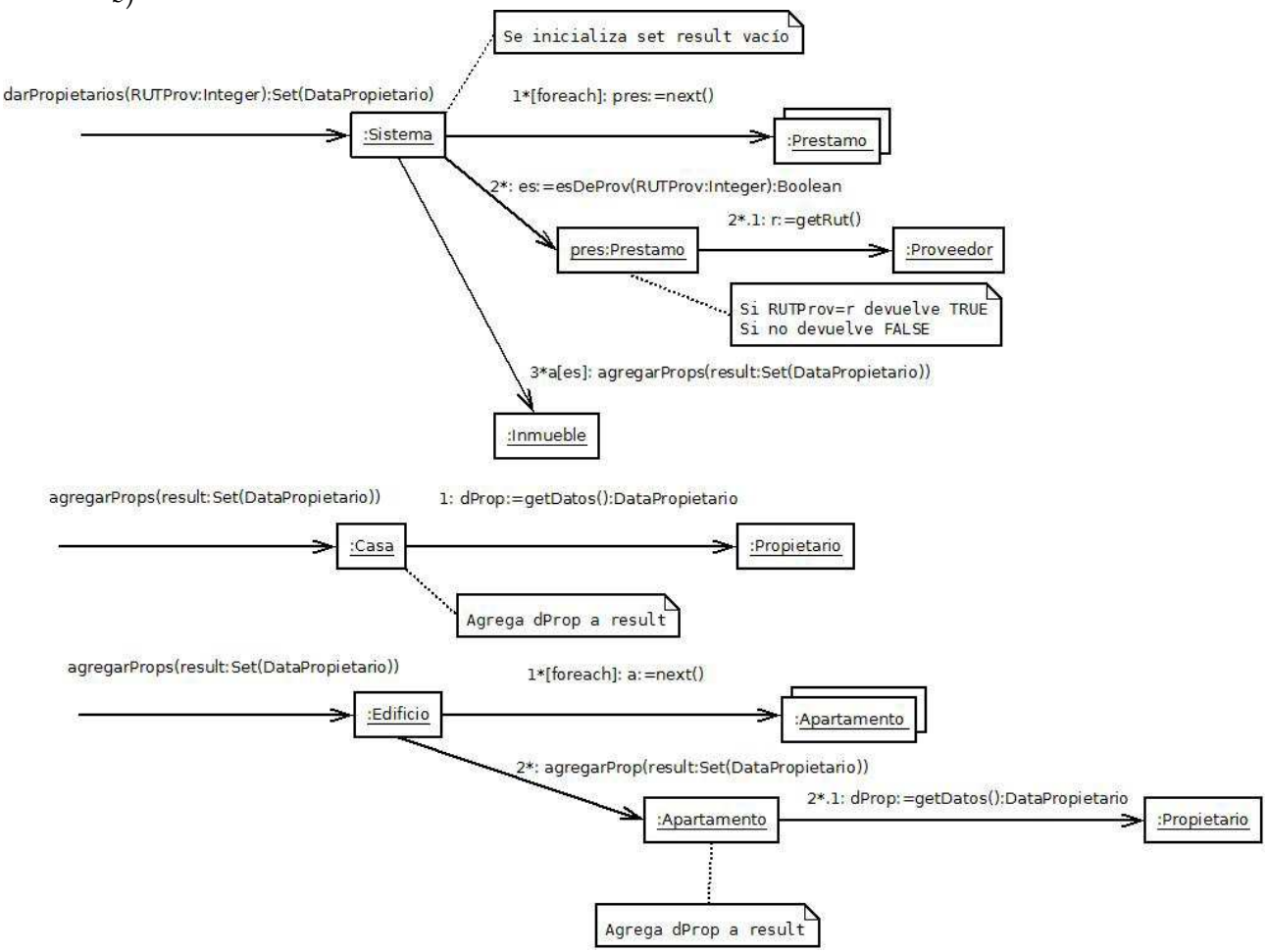
Obs: debido al software utilizado, se debieran eliminar los números de mensajes (puesto que no son necesarios), así como los bloques de activación de mensajes (los bloques verticales blancos) así como agregar la línea de separación entre ambas partes del frame ALT (línea punteada horizontal) y colocar el datatype DTSesion.

Problema 2

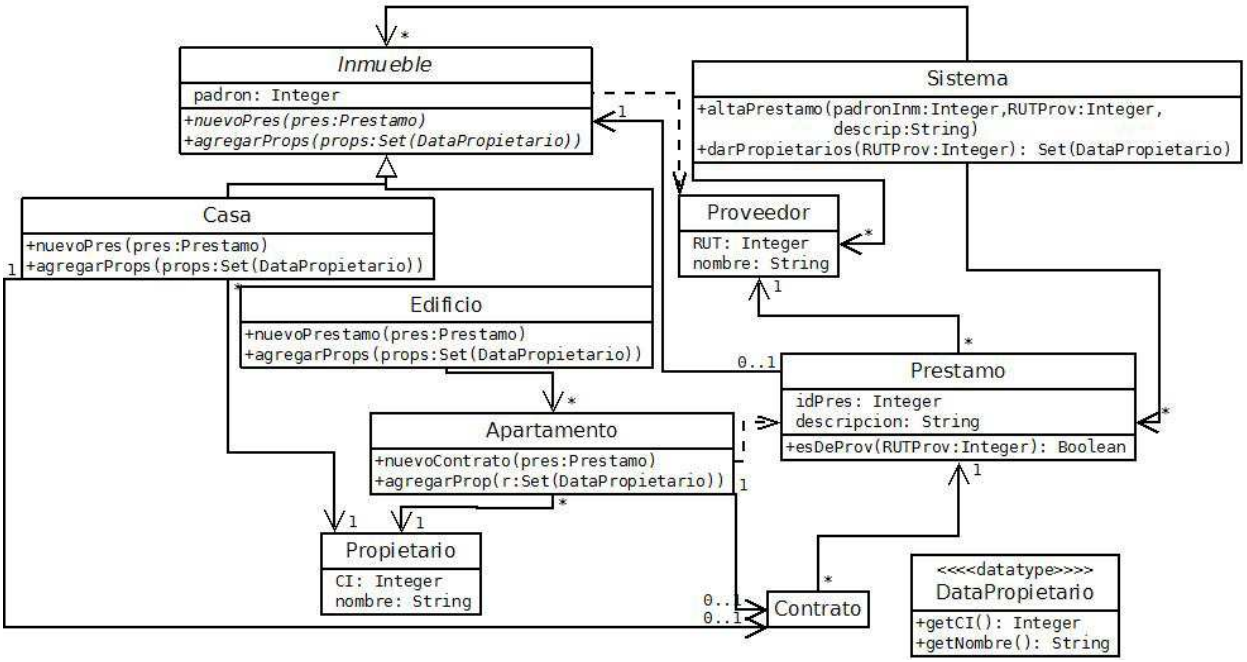
a)


altaPrestamo(padronInm,RUTProv:Integer;descrip:String)
:Sistema
1: i:=find(padronInm:Integer)
:Inmueble
2: p:=find(RUTProv:Integer)
:Proveedor
3: pres:=create(i:Inmueble;p:Proveedor;descrip:String;idPres:Integer)
pres:Prestamo
3.1: nuevoPres(self:Prestamo)
:Inmueble
4: add(pres:Prestamo)
:Prestamo
idPres es autogenerado por el Sistema
Se mantiene referencia a i y p
nuevoPres(pres:Prestamo)
:Casa
1: c:=create(pres:Prestamo)
:Contrato
Se mantiene referencia a pres
Se mantiene referencia a c
nuevoPres(pres:Prestamo)
:Edificio
1*[foreach]: a:=next()
:Apartamento
2*: nuevoContrato(pres:Prestamo)
:Apartamento
2*.1: c:=create(pres:Prestamo)
:Contrato
Se mantiene referencia a c
Se mantiene referencia a pres

b)


Se inicializa set result vacío
darPropietarios(RUTProv:Integer):Set(DataPropietario)
1*[foreach]: pres:=next()
:Sistema
:Prestamo
2*: es:=esDeProv(RUTProv:Integer):Boolean
2*.1: r:=getRut()
pres:Prestamo
:Proveedor
Si RUTProv=r devuelve TRUE
Si no devuelve FALSE
3*a[es]: agregarProps(result:Set(DataPropietario))
:Inmueble
agregarProps(result:Set(DataPropietario))
1: dProp:=getDatos():DataPropietario
:Casa
:Propietario
Agrega dProp a result
agregarProps(result:Set(DataPropietario))
1*[foreach]: a:=next()
:Edificio
:Apartamento
2*: agregarProp(result:Set(DataPropietario))
2*.1: dProp:=getDatos():DataPropietario
:Apartamento
:Propietario
Agrega dProp a result

c)


Inmueble
padron: Integer
+nuevoPres(pres:Prestamo)
+agregarProps(props:Set(DataPropietario))
Sistema
+altaPrestamo(padronInm:Integer,RUTProv:Integer, descrip:String)
+darPropietarios(RUTProv:Integer): Set(DataPropietario)
Casa
+nuevoPres(pres:Prestamo)
+agregarProps(props:Set(DataPropietario))
Proveedor
RUT: Integer
nombre: String
Edificio
+nuevoPrestamo(pres:Prestamo)
+agregarProps(props:Set(DataPropietario))
Prestamo
idPres: Integer
descripcion: String
+esDeProv(RUTProv:Integer): Boolean
Apartamento
+nuevoContrato(pres:Prestamo)
+agregarProp(r:Set(DataPropietario))
Propietario
CI: Integer
nombre: String
Contrato
<<<<datatype>>>>
DataPropietario
+getCI(): Integer
+getNombre(): String

Problema 3

```
/****************** USUARIO *********************/

//Usuario.h
class Usuario {
private:
	int id;
	string usuario;
	string email;
public:
	Usuario();
	void invitar(int idRemitente, DataJuego dataJuego);
	virtual ~Usuario();
};

//Usuario.cpp
Usuario::Usuario() {}

Usuario::~Usuario() {}

void Usuario::invitar(int idRemitente, DataJuego dataJuego) {
	//método vacío
}

/********************** AJEDREZ **************************/

//Ajedrez.h
class Ajedrez : public Juego {
public:
	Ajedrez();
	virtual Juego* clonar();
	virtual DataResultado comando(int, string);
	virtual ~Ajedrez();
};

//Ajedrez.cpp
Ajedrez::Ajedrez() {
}

Juego* Ajedrez::clonar() {
	return new Ajedrez();
}
```

```
DataResultado Ajedrez::comando(int, string) {
	DataResultado dr = new DataResultado();
	//metodo vacío
	return dr;
}


Ajedrez::~Ajedrez() {}

/************************ DATAJUEGO ******************************/

//DataJuego.h
class DataJuego {
private:
	int id;
	string nombre;
	string descripcion;
	string ayuda;

public:
	DataJuego(int, string, string, string);
	virtual ~DataJuego();
	string DataJuego::getAyuda();
	string DataJuego::getDescripcion();
	string DataJuego::getNombre();
	int DataJuego::getId();
};


//DataJuego.cpp
DataJuego::DataJuego(int idjuego, string nom, string desc, string ayu) {
	id = idjuego;
	nombre = nom;
	descripcion = desc;
	ayuda = ayu;
}

DataJuego::~DataJuego() {
}

string DataJuego::getAyuda() {
	return ayuda;
}
```

```
string DataJuego::getDescripcion() {
	return descripcion;
}

int DataJuego::getId() {
	return id;
}

string DataJuego::getNombre() {
	return nombre;
}


/********************* JUEGO ***************************/

//Juego.h
class Juego {
protected:
	int id;
public:
	Juego();
	virtual Juego* clonar() = 0;
	virtual ~Juego() = 0;
	virtual DataResultado comando(int idUsuario, string comando) = 0;
};

//Juego.cpp
Juego::Juego() {}

Juego::~Juego() {}

/************************ MANEJADORJUEGOS ***************************/

//ManejadorJuegos.h
class ManejadorJuegos {
private:
	ManejadorJuegos();
	int ultimoId;
	static ManejadorJuegos* instancia;
	map<string, Juego*> prototipos;
	map<int, Juego*> juegos;
public:
```

```
	Juego* nuevoJuego(string nombre);
	Juego* getJuego(int idJuego);
	static ManejadorJuegos* getInstancia();
	virtual ~ManejadorJuegos();
};

//ManejadorJuegos.cpp
ManejadorJuegos::ManejadorJuegos() {
	ultimoId = 0;
	prototipos["Ajedrez"] = new Ajedrez();
	prototipos["Ludo"] = new Ludo();
}

ManejadorJuegos* ManejadorJuegos::instancia = NULL;

Juego* ManejadorJuegos::nuevoJuego(string nombre) {
	Juego* nuevo = prototipos[nombre]->clonar();
	int id = ++ultimoId;
	nuevo->setId(id);
	juegos[id] = nuevo;

}

Juego* ManejadorJuegos::getJuego(int idJuego) {
	return juegos[idJuego];
}

ManejadorJuegos* ManejadorJuegos::getInstancia() {
	if (instancia == NULL) {
		instancia = new ManejadorJuegos();
	}
	return ManejadorJuegos();
}

ManejadorJuegos::~ManejadorJuegos() {
	map<int, Juego*>::iterator it;
	for (it = juegos.begin(); it != juegos.end(); it++) {
		delete it->second;
	}
	for (it = prototipos.begin(); it != prototipos.end(); it++) {
		delete it->second;
	}
}
```

```
/********************* SERVIDORJUEGOS ***************************/

//ServidorJuegos.h
class ServidorJuegos {
private:
	static ServidorJuegos* instancia;
	map<int, Usuario*> usuarios;
	ServidorJuegos();
public:
	static ServidorJuegos* getInstancia();
	DataJuego nuevoJuego(int idUsuario, int idInvitados[], string
nombreJuego);
	DataResultado nuevoComando(int idUsuario, int idJuego, string comando);
	virtual ~ServidorJuegos();
};

//ServidorJuegos.cpp
ServidorJuegos* ServidorJuegos::instancia = NULL;

ServidorJuegos::ServidorJuegos() {
}

ServidorJuegos::~ServidorJuegos() {
	map<int,Usuario*>::iterator it;
	for(it = usuarios.begin(); it != usuarios.end(); it++) {
		delete it->second;
	}
}

ServidorJuegos* ServidorJuegos::getInstancia() {
	if (instancia == NULL) {
		instancia = new ServidorJuegos();
	}
	return instancia;
}

DataJuego ServidorJuegos::nuevoJuego(int idUsuario, int idInvitados[],
		string nombreJuego) {
	ManejadorJuegos* mj = ManejadorJuegos::getInstancia();
	Juego* juego = mj->nuevoJuego(nombreJuego);
	DataJuego dj = juego->getData();
	int cantInvitados = sizeof(idInvitados) / sizeof(int);
```

```
	for (int i = 0; i < cantInvitados; i++) {
		usuarios[idInvitados[i]]->invitar(idUsuario, dj);
	}
	return dj;
}

DataResultado ServidorJuegos::nuevoComando(int idUsuario, int idJuego,
		string comando) {
	Juego* juego = ManejadorJuegos::getInstancia()->getJuego(idJuego);
	return juego->comando(idUsuario, comando);
}
```