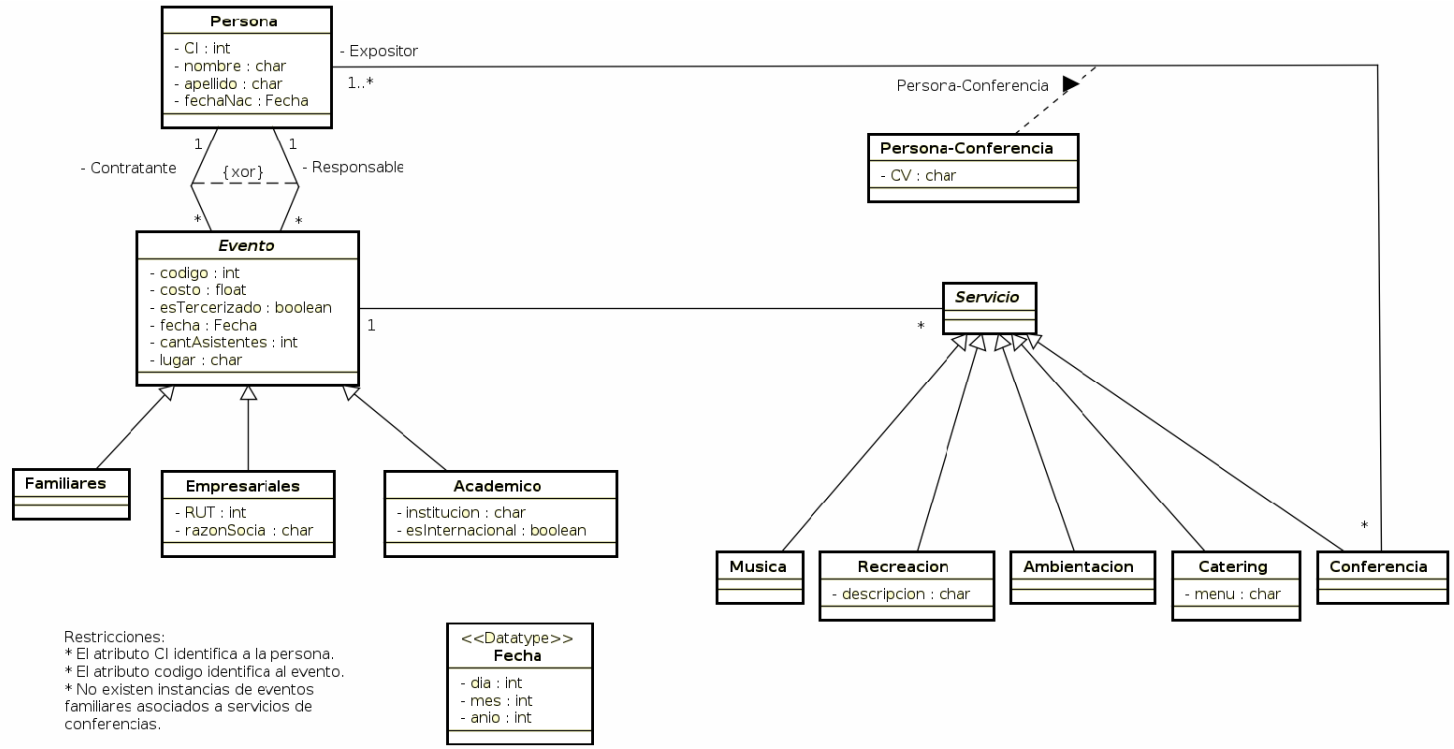
Programación Avanzada

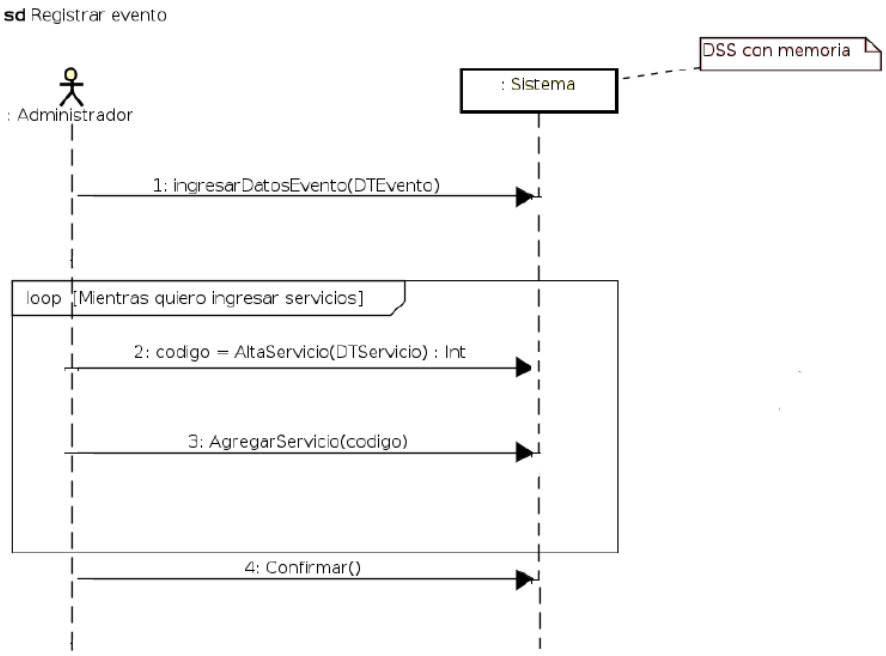
SOLUCIÓN EXAMEN JULIO 2018

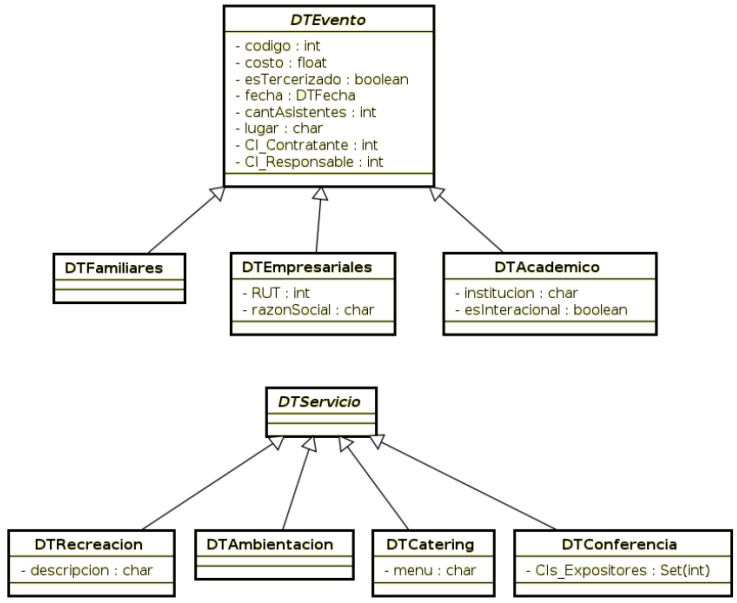
Problema 1 (34 puntos)

a) Modelo de Dominio:



b) DSS Registrar Evento:





Problema 2 (34 puntos)

a) Diagrama de Comunicación "*Aprender Habilidad Especial*"

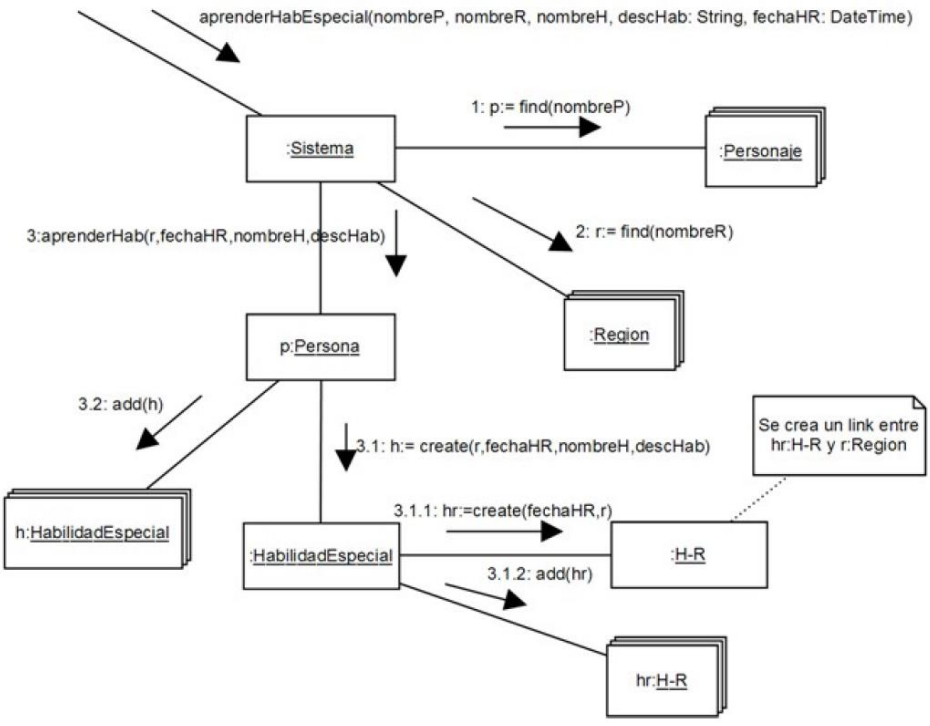
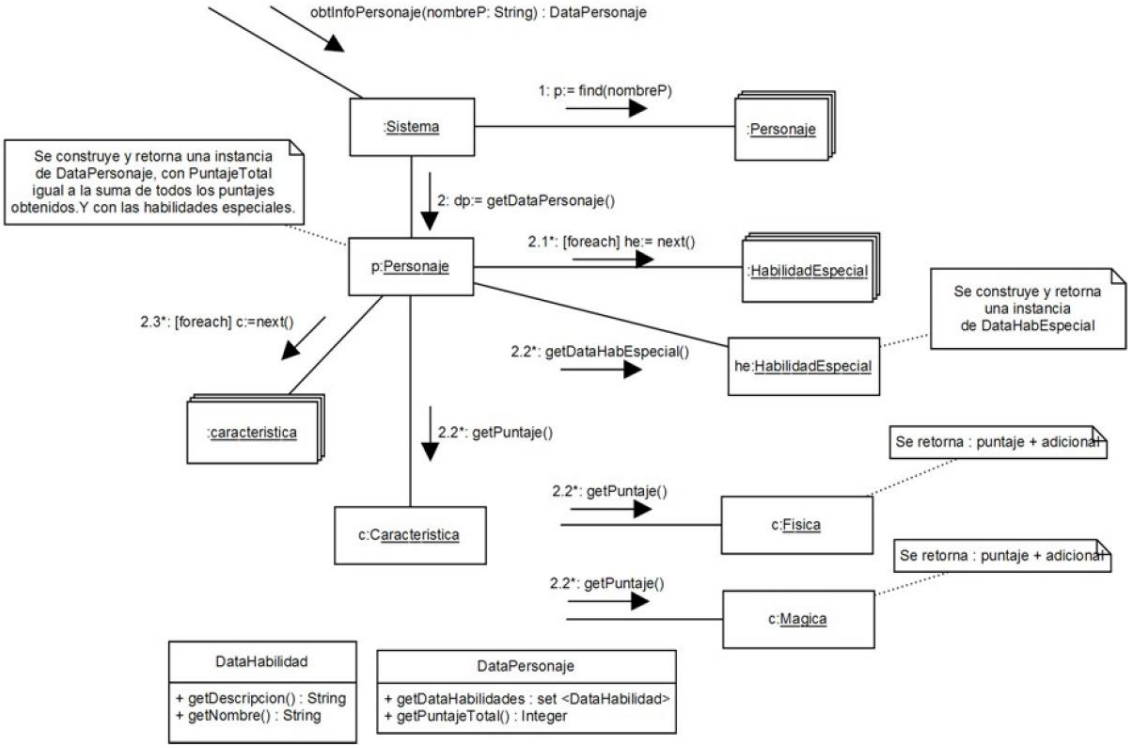
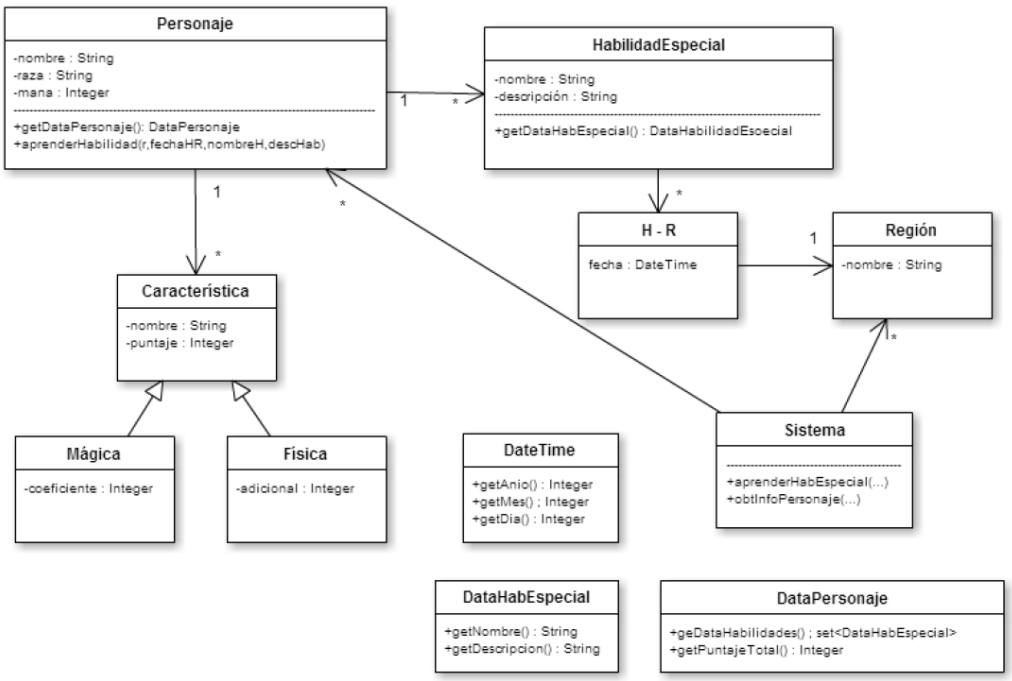


Diagrama de Comunicación "*Obtener información de un personaje*"



b) Diagrama de Clases de Diseño:



Problema 3 (33 puntos)

```
/****************** 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 getAyuda();
		string getDescripcion();
		string 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);
}
```