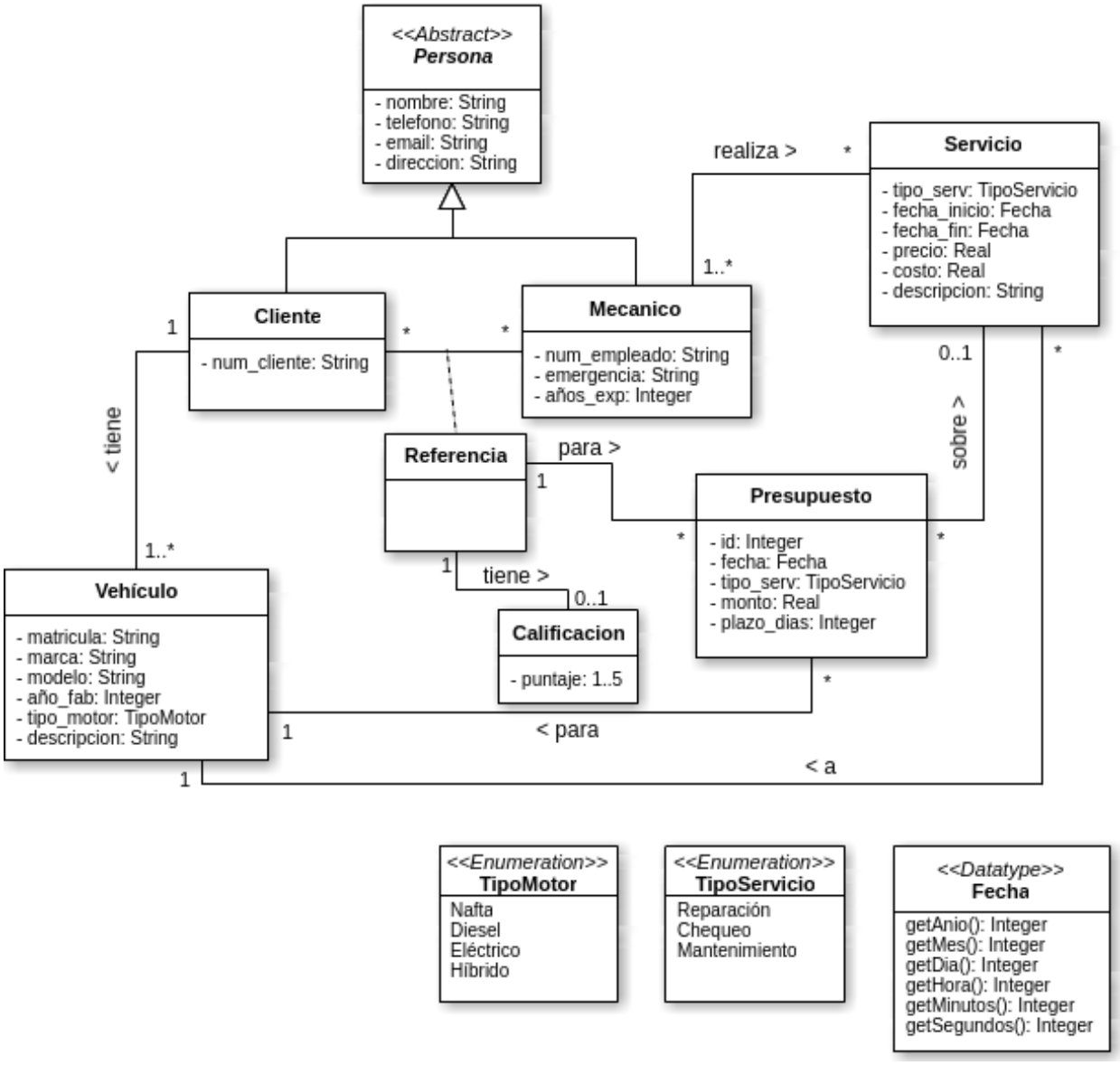
Programación 4

EXAMEN JULIO 2016

SOLUCION

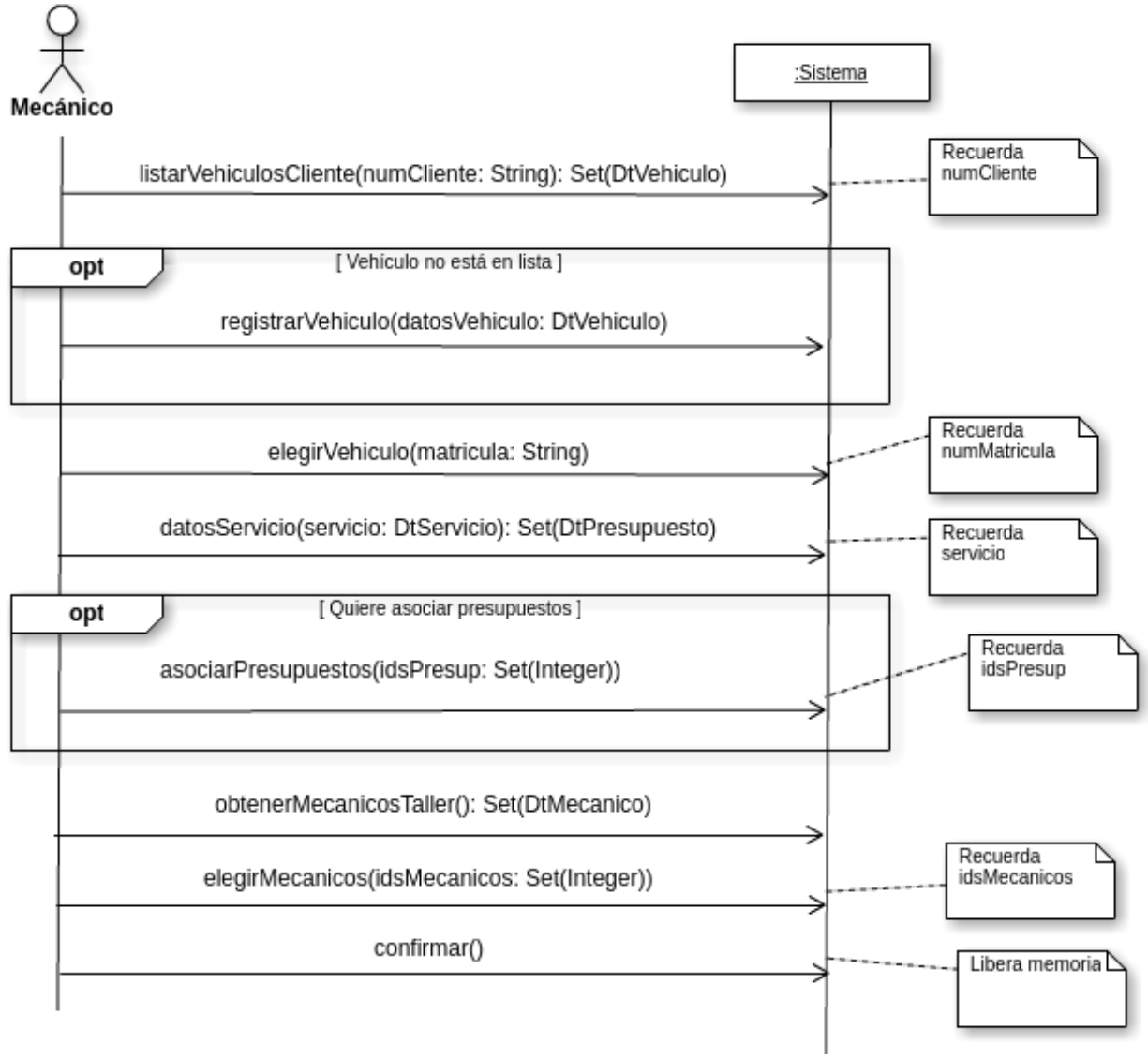
<u>Problema 1</u> (35 puntos)

i)



No existen dos Clientes con mismo valor de num_cliente.
No existen dos Mecánicos con mismo valor de num_empleado.
No existen dos Vehículos con mismo valor de matrícula.
No existen dos Presupuestos con mismo valor de id.
Dado un Presupuesto p, debe relacionarse únicamente con un Vehículo del Cliente relacionado con p a través de una instancia de Referencia.
Dados un Presupuesto p relacionado a un Servicio s, a su vez estos dos se relacionan con el mismo Vehículo v.
Un Presupuesto solo puede relacionarse con un Servicio si los valores en sus respectivos atributos tipo_serv coinciden.

ii)


Mecánico
:Sistema
listarVehiculosCliente(numCliente: String): Set(DtVehiculo)
Recuerda numCliente
opt
[Vehículo no está en lista]
registrarVehiculo(datosVehiculo: DtVehiculo)
elegirVehiculo(matricula: String)
Recuerda numMatricula
datosServicio(servicio: DtServicio): Set(DtPresupuesto)
Recuerda servicio
opt
[Quiere asociar presupuestos]
asociarPresupuestos(idsPresup: Set(Integer))
Recuerda idsPresup
obtenerMecanicosTaller(): Set(DtMecanico)
elegirMecanicos(idsMecanicos: Set(Integer))
Recuerda idsMecanicos
confirmar()
Libera memoria

Problema 2 (30 puntos)

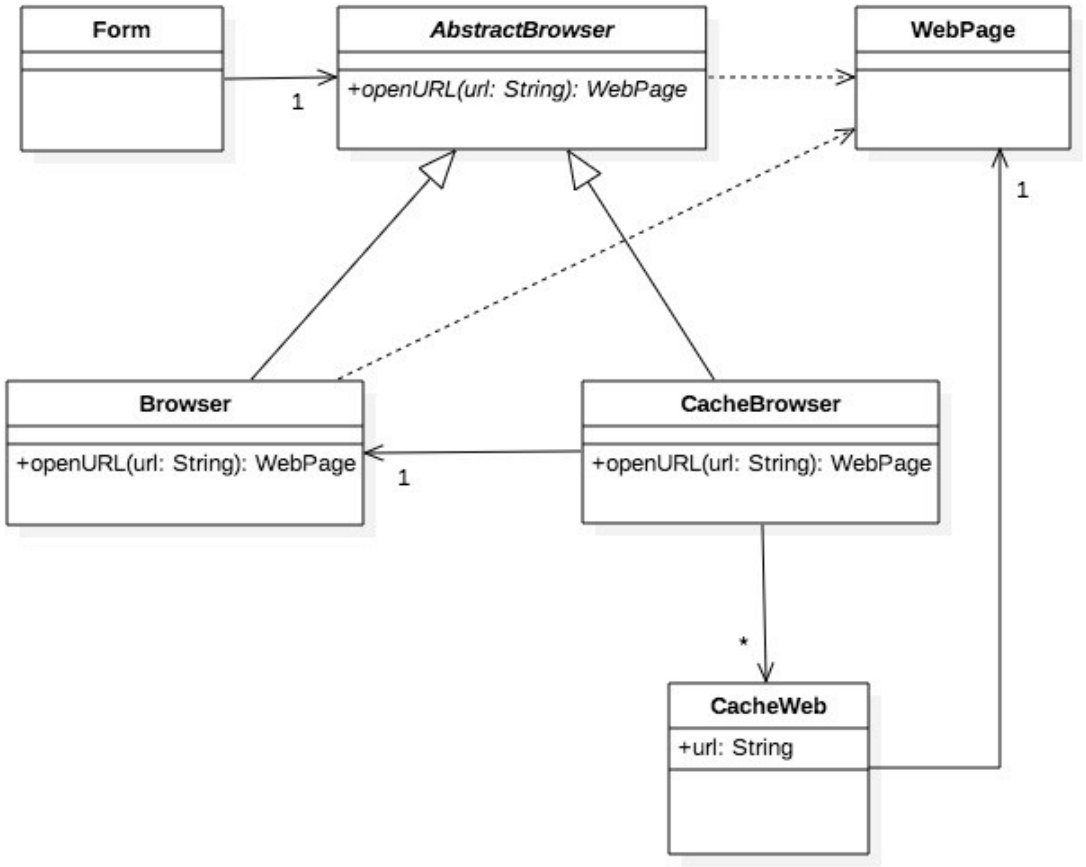
Parte a)
i) Las PRE como las POST condiciones establecen el estado del Sistema ANTES y DESPUES de ejecutar la operacion, y se describen en términos de: 1) Creación de Objetos; 2) Destrucción de Objetos; 3) Creación de Links; 4) Destrucción de Links; y 5) Modificación de Valores de Atributos. Además, las PRE pueden describir los parámetros de las operaciones, mientras que las POST el valor de retorno.

ii) Se necesita una Fabrica para poder instanciar Controladores desde la Presentación sin conocerlos, por lo tanto, sin quedar acoplados a ellos. Por esto se denominan Fabricas de Controladores.

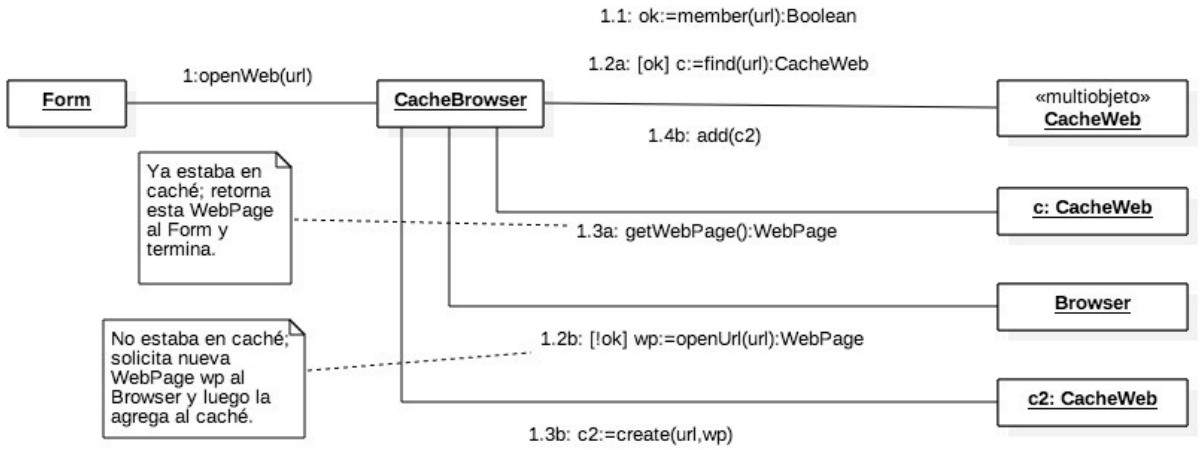
Parte b)
i) Se propone utilizar el patrón Proxy de forma que el Formulario siga solicitando paginas web al Browser desconociendo la existencia del sistema de cache, permitiendo este patrón no modificar la clase Browser.
Los participantes del Proxy en este caso son:

- Real Subject: Browser
- Proxy: CacheBrowser
- Subject: AbstractBrowser
- Client: Formulario

ii)



iii)


1.1: ok:=member(url):Boolean
1.2a: [ok] c:=find(url):CacheWeb
1:openWeb(url)
Form
CacheBrowser
«multiobjeto»
CacheWeb
1.4b: add(c2)
Ya estaba en caché; retorna esta WebPage al Form y termina.
c: CacheWeb
1.3a: getWebPage():WebPage
Browser
No estaba en caché; solicita nueva WebPage wp al Browser y luego la agrega al caché.
1.2b: [!ok] wp:=openUrl(url):WebPage
c2: CacheWeb
1.3b: c2:=create(url,wp)

Problema 3 (35 puntos)

i. Los patrones utilizados son:

- State, donde Climatizador es el contexto, ModoClim es el estado y Frio y Calor son los estados concretos.
- Strategy, donde calor es el contexto, DispCalor es la estrategia y Electrico y Gas son las estrategias concretas.

ii.

```
class Climatizador {
      private:
            int tempActual;
            ModoClim *modoActual;
      public:
            void cambiarModo();
            void subirTemp();
            void bajarTemp();
};

void Climatizador::cambiarModo() {
      ModoClim *modoAnterior = modoActual;
      modoActual = modoActual->darSigModo();
      delete modoAnterior;
}

void Climatizador::subirTemp() {
      if (tempActual + 1 > Utils::maxTemp)
            throw invalid_argument("No se puede establecer temperatura tan
alta");
      modoActual->setTemp(tempActual + 1);
      tempActual = tempActual + 1;
}

void Climatizador::bajarTemp() {
      if (tempActual - 1 > Utils::minTemp)
            throw invalid_argument("No se puede establecer temperatura tan
baja");
      modoActual->setTemp(tempActual - 1);
      tempActual = tempActual - 1;
}

class ModoClim {
      public:
            virtual ModoClim *darSigModo() = 0;
            virtual void setTemp(int) = 0;
};

class Frio : public ModoClim {
      private:
            DispFrio *disp;
      public:
            ModoClim *darSigModo();
            void setTemp(int);
};

ModoClim *Frio::darSigModo() {
      return new Calor();
}
```

```
void Frio::setTemp(int t) {
      disp->setTemp(t);
}

class Calor : public ModoClim {
      private:
            DispCalor *actual;
            map<int,DispCalor *> catalogo;
      public:
            ModoClim *darSigModo();
            void setTemp(int);
            void revisarPlan();
            void agregarDisp(DispCalor *);
            void eliminarDisp(int)
};

ModoClim *Calor::darSigModo() {
      return new Frio();
}

void Calor::setTemp(int t) {
      actual->setTemp(t);
}

void Calor::revisarPlan() {
      DispCalor *nuevo = catalogo[Utils::darDispCalor()];
      actual = nuevo;
}

void Calor::agregarDisp(DispCalor *d) {
      catalogo[d->getId()] = d;
}

void Calor::eliminarDisp(int id) {
      catalogo.erase(id);
}

class DispCalor {
      private:
            int id;
      public:
            virtual void setTemp(int) = 0;
            int getId();
};

int DispCalor::getId() {
      return id;
}
```