Programación 4

EXAMEN FEBRERO 2007

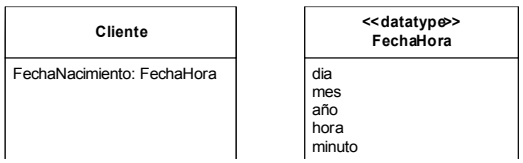
SOLUCION

Problema 1 (25 puntos)

a)

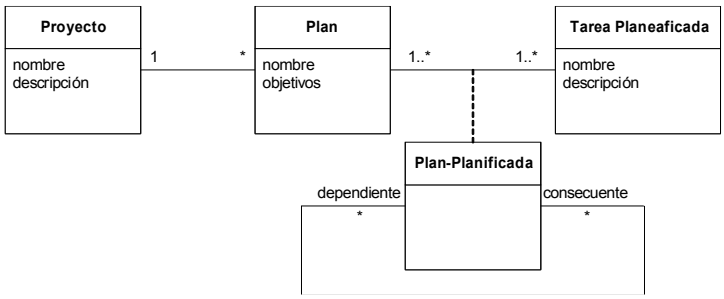
i. Ver slides de teórico: 09 – Análisis – Modelado de dominio slides 10 - 13.

ii.



b)

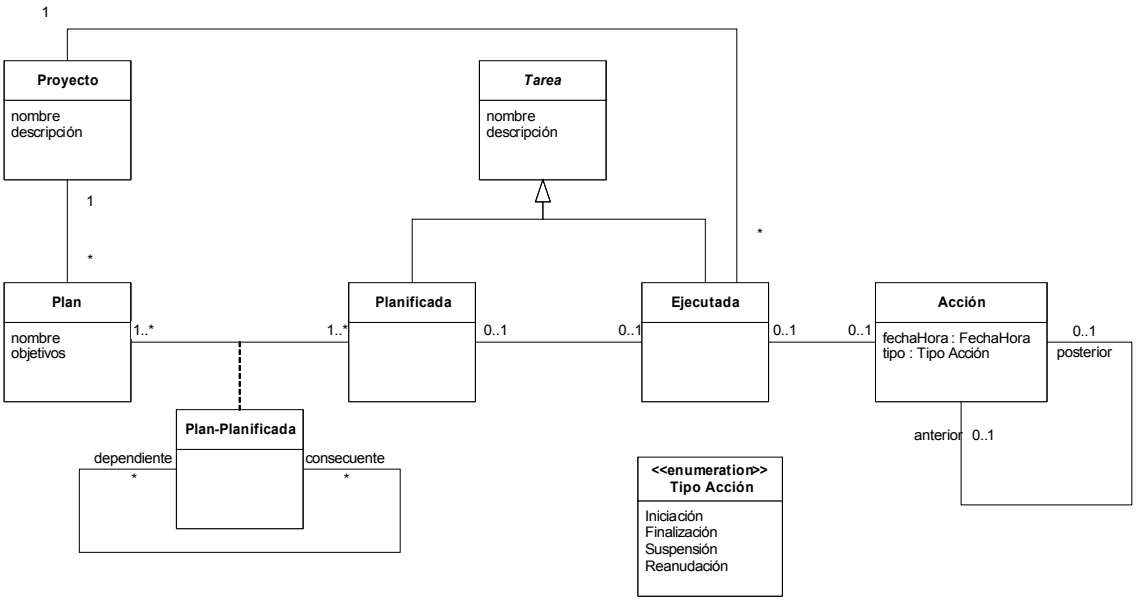
i.



Restricciones:

```
context Proyecto
-- El nombre identifica el proyecto.
inv: Proyecto.allInstances()->isUnique(nombre)

context Plan
-- El nombre identifica al plan.
inv: Plan.allInstances()->isUnique(nombre)

context Tarea
-- El nombre identifica la tarea.
inv: Tarea.allInstances()->isUnique(nombre)

context Plan-Planificada
-- Las tareas de un plan sólo pueden estar relacionadas con otras
-- tareas del mismo plan.
inv: self.consecuente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Una tarea no puede depender de sí misma
inv: self.consecuente->excludes(self)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->excludes(self)
```

ii.



Restricciones:

```
context Proyecto
-- El nombre identifica el proyecto.
inv: Proyecto.allInstances()->isUnique(nombre)

context Plan
-- El nombre identifica al plan.
inv: Plan.allInstances()->isUnique(nombre)

context Tarea
-- El nombre identifica la tarea.
inv: Tarea.allInstances()->isUnique(nombre)

context Plan-Planificada
-- Las tareas de un plan sólo pueden estar relacionadas con otras
-- tareas del mismo plan.
inv: self.consecuente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->forAll(p : Plan-Planificada | self.plan = p.plan)

-- Una tarea no puede depender de sí misma
inv: self.consecuente->excludes(self)

-- Análogo para los dependientes (no requerida, se pone a modo de aclaración)
inv: self.dependiente->excludes(self)

context Acción
-- Para cada acción, la fecha de la acción posterior debe ser mayor o igual
inv: self.posterior.fecha >= self.fecha

-- Una tarea iniciada puede ser suspendida o finalizada
inv: self.tipo = TipoAcción::Iniciación implies (self.posterior.tipo =
TipoAcción::Finalización or self.posterior.tipo = TipoAcción::Suspensión)

-- Una tarea suspendida sólo puede ser reanudada
inv: self.tipo = TipoAcción::Suspensión implies self.posterior.tipo =
TipoAcción::Reanudación

-- Una tarea finalizada no puede sufrir acciones posteriores
inv: self.tipo = TipoAcción::Suspensión implies self.posterior->isEmpty()
```

Problema 2 (25 puntos)

a)

i. Explique la relación entre caso de uso, escenario y diagrama de secuencia del sistema.
Para un caso de uso pueden existir varios escenarios (tipicos y alternativos). Cada escenario de un caso de uso se puede representar con un DSS.

ii. Explique la relación entre caso de uso y colaboración.
Una colaboración realiza uno o varios casos de uso.

iii. Describa los 2 enfoques citados en el curso para diseñar una colaboración.
- Definir primero la estructura y luego generar las diferentes interacciones respetando dicha estructura.
- Definir las interacciones (teniendo en cuenta los criterios GRASP) y luego definir la estructura necesaria para que dichas interacciones puedan ocurrir.

b) Se quiere terminar el análisis y realizar el diseño de una iteración de un sistema de gestión de solicitudes de tarjetas y registro de movimientos usando las mismas. A continuación se presentan el modelo conceptual y los casos de usos (junto con un diagrama de secuencia del sistema del curso típico de eventos) de esta iteración.

i. Contratos

```
void otorgar(cedula : String, nro_tarjeta : String)
pre: existe una instancia de Persona con cédula de identidad = cedula
pre: existe una instancia de Tarjeta identificada por el número = nro_tarjeta
pre: no existe un link de Otorgamiento entre la instancia de Persona con cédula de identidad = cedula y la instancia de Tarjeta identificada por el número = nro_tarjeta

post: si existe un link de Solicitud entre la instancia de Persona con cédula de identidad = cedula y la instancia de Tarjeta identificada por el número = nro_tarjeta, entonces se elimina dicho link y se crea un link de Otorgamiento entre el mismo par de instancias.
```

bool inicioMovs(cedula : String, nro_tarjeta : String)
pre: existe una instancia de Persona con cédula de identidad = cedula
pre: existe una instancia de Tarjeta identificada por el número = nro_tarjeta

post: el sistema crea un link a la instancia de Tarjeta identificada por nro_tarjeta como la tarjeta actual
post: el sistema mantiene en memoria la cantidad ingresada y le asigna el valor 0
post: devuelve true si existe un link de Otorgamiento entre la instancia de Persona con cédula de identidad = cedula y la instancia de Tarjeta identificada por el número = nro_tarjeta. En caso contrario, devuelve false.

-- Notar que ni en el modelo conceptual ni en los casos de usos se destaca una relación entre el precioUnitMov y el precioActual del Producto.

void registrarMov(m : MovDT)
pre: el sistema tiene una instancia de Tarjeta como tarjeta actual

post: se crea una instancia de Movimiento con valor de atributo fecha = m.getFecha(), cant = m.getCantidad() y precioUnitMov = m.getPrecio()
post: se crea un link entre la tarjeta actual recordada por el sistema y la instancia de Movimiento creada
post: se crea un link entre la instancia de Movimiento creada y la instancia de Producto identificada por el código = m.getCodigoProducto() y nombre de empresa = m.getNombreEmpresa()
post: si y sólo si no existe un instancia de Producto identificada por el código = m.getCodigoProducto() y nombre de empresa = m.getNombreEmpresa(), se crea una instancia de Producto con valor de atributo codigo = m.getCodigoProducto(), nombreEmpresa = m.getNombreEmpresa(), descripcion = m.getDescripcion() y precioActual = m.getPrecio()
post: se incrementa en 1 la cantidad ingresada

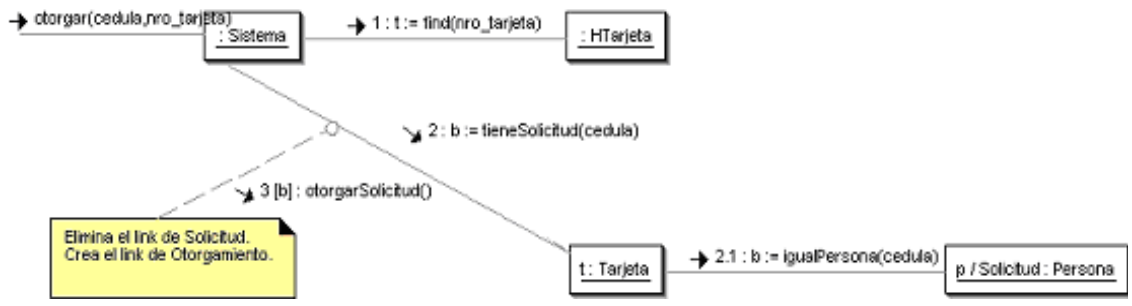
int finMovs()
pre: el sistema tiene una instancia de Tarjeta como tarjeta actual

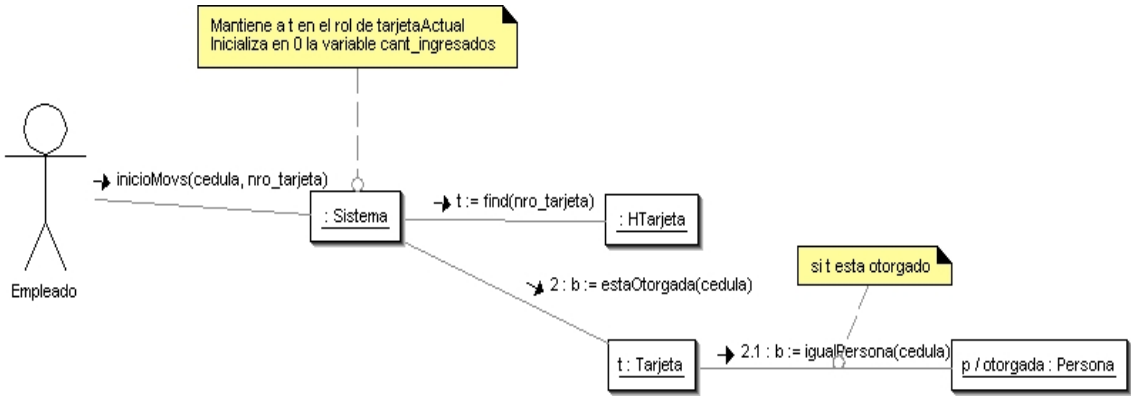
post: el sistema elimina el link a la instancia de Tarjeta identificada por nro_tarjeta como la tarjeta actual
post: devuelve como resultado el valor de cantidad ingresada

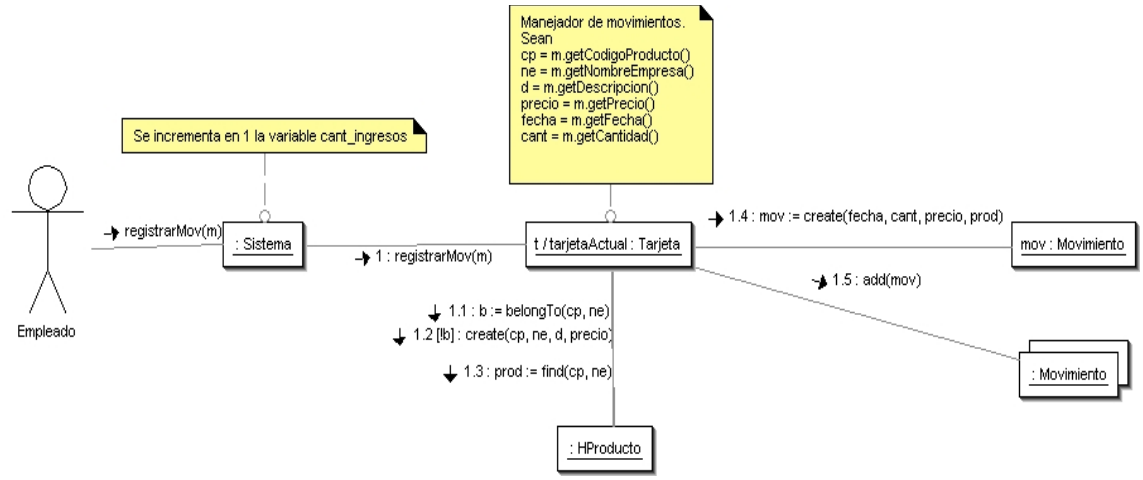
void eliminarTarjeta(nro_tarjeta : String)
pre: existe una instancia de Tarjeta identificada por el número = nro_tarjeta

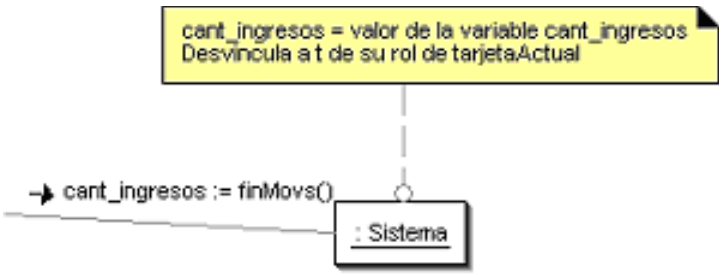
post: no existe instancia de Tarjeta identificada por el número = nro_tarjeta
post: no existen instancias de Movimiento que estaban asociadas con la
instancia de Tarjeta identificada por el número = nro_tarjeta
post: se eliminan los links que existían entre las instancias de producto y las instancias de Movimiento que estaban asociadas con la
instancia de Tarjeta identificada por el número = nro_tarjeta
post: si existía link de Solicitud entre la instancia de Tarjeta identificada por el número = nro_tarjeta y una instancia de Persona, se lo elimina
post: si existía link de Otorgamiento entre la instancia de Tarjeta identificada por el número = nro_tarjeta y una instancia de Persona, se lo elimina

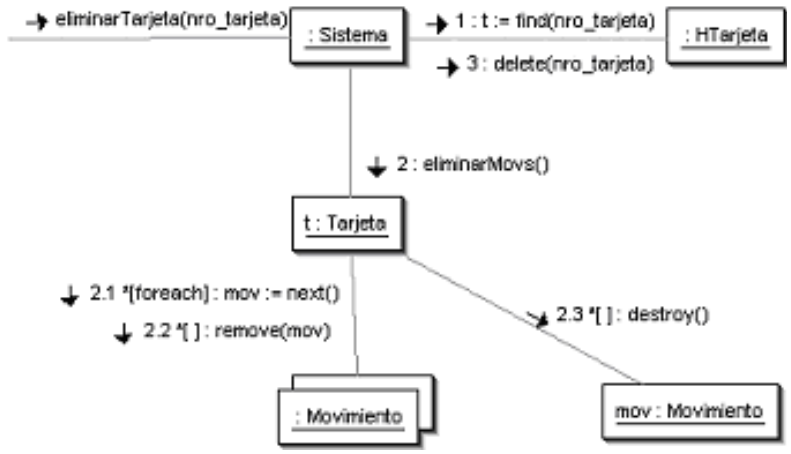
ii. Diagramas de Comunicación






Manejador de movimientos.
Sean
cp = m.getCodigoProducto()
ne = m.getNombreEmpresa()
d = m.getDescripcion()
precio = m.getPrecio()
fecha = m.getFecha()
cant = m.getCantidad()
Se incrementa en 1 la variable cant_ingresos
Empleado
registrarMov(m)
: Sistema
1 : registrarMov(m)
t / tarjetaActual : Tarjeta
1.4 : mov := create(fecha, cant, precio, prod)
mov : Movimiento
1.5 : add(mov)
1.1 : b := belongTo(cp, ne)
1.2 [!b] : create(cp, ne, d, precio)
1.3 : prod := find(cp, ne)
: Movimiento
: HProducto

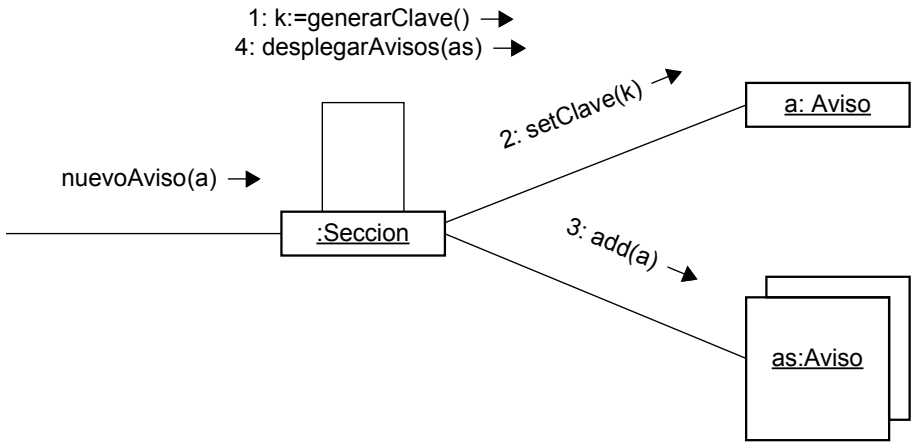

cant_ingresos = valor de la variable cant_ingresos
Desvincula a t de su rol de tarjetaActual
cant_ingresos := finMovs()
: Sistema

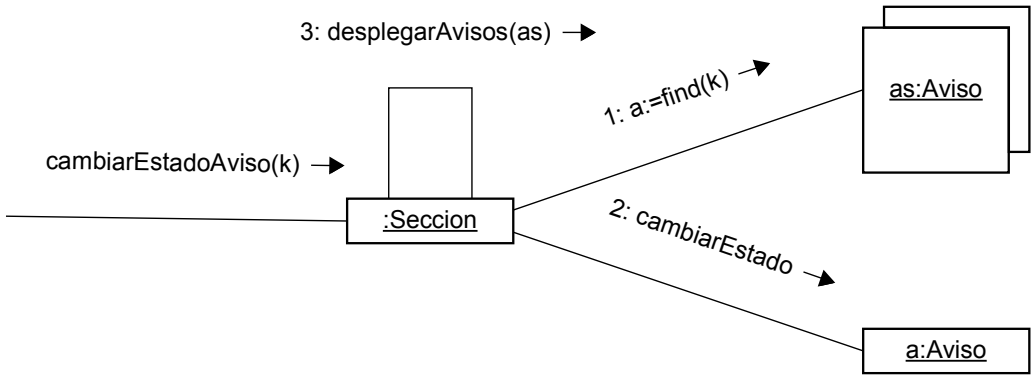

eliminarTarjeta(nro_tarjeta)
: Sistema
1 : t := find(nro_tarjeta)
: HTarjeta
3 : delete(nro_tarjeta)
2 : eliminarMovs()
t : Tarjeta
2.1 *[foreach] : mov := next()
2.2 *[ ] : remove(mov)
2.3 *[ ] : destroy()
: Movimiento
mov : Movimiento

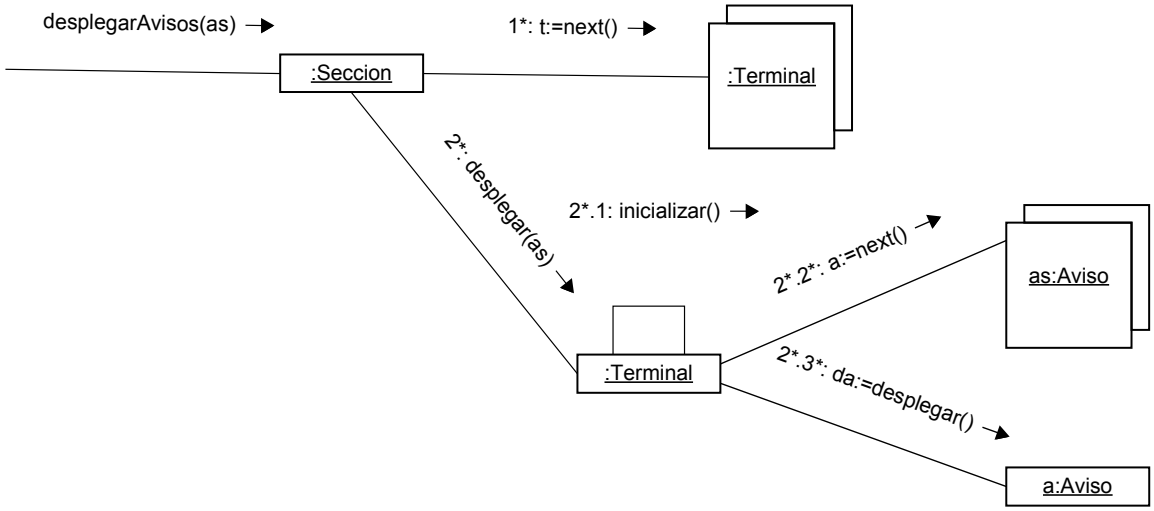
Problema 3 (25 puntos)

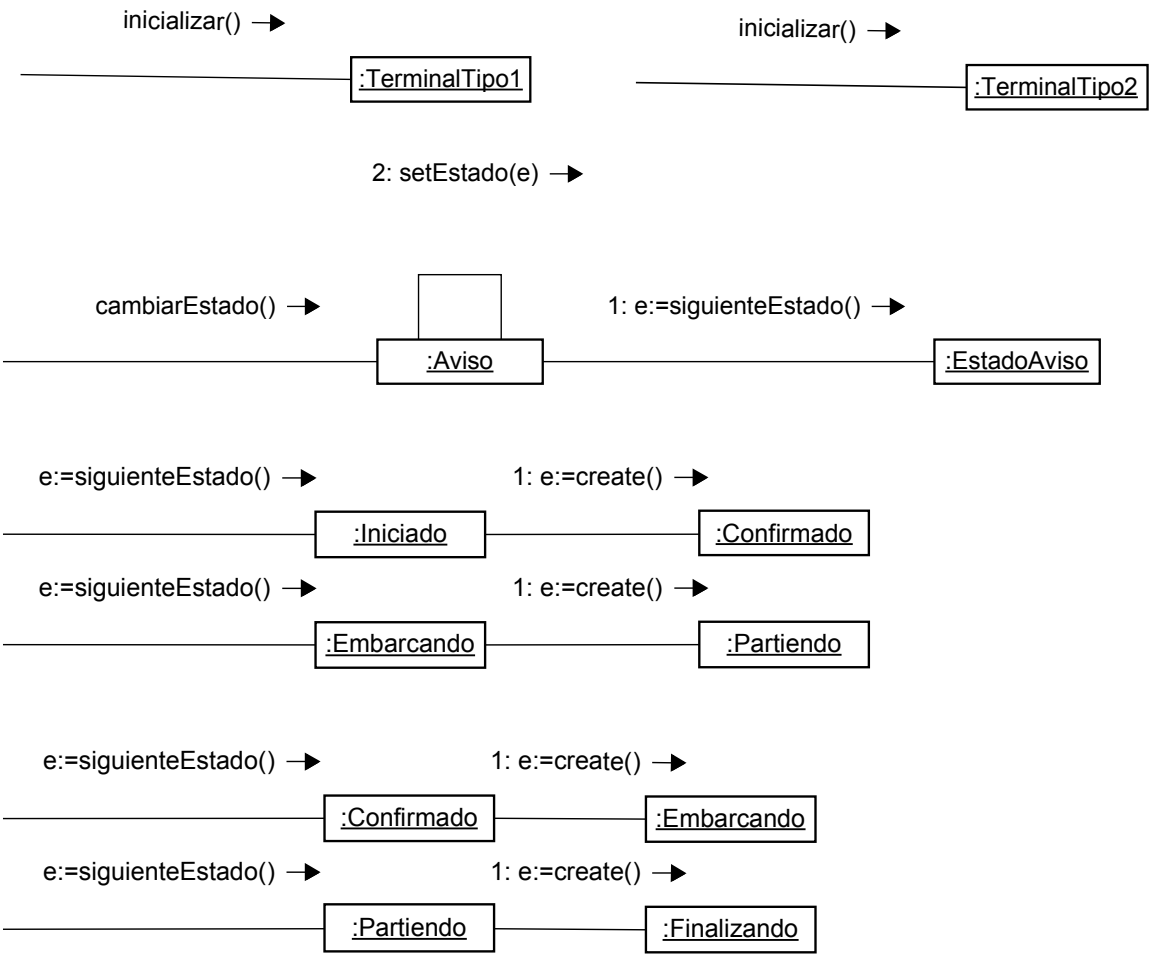
a) Ver teórico diseño, diagramas de comunicación.

b)
i.

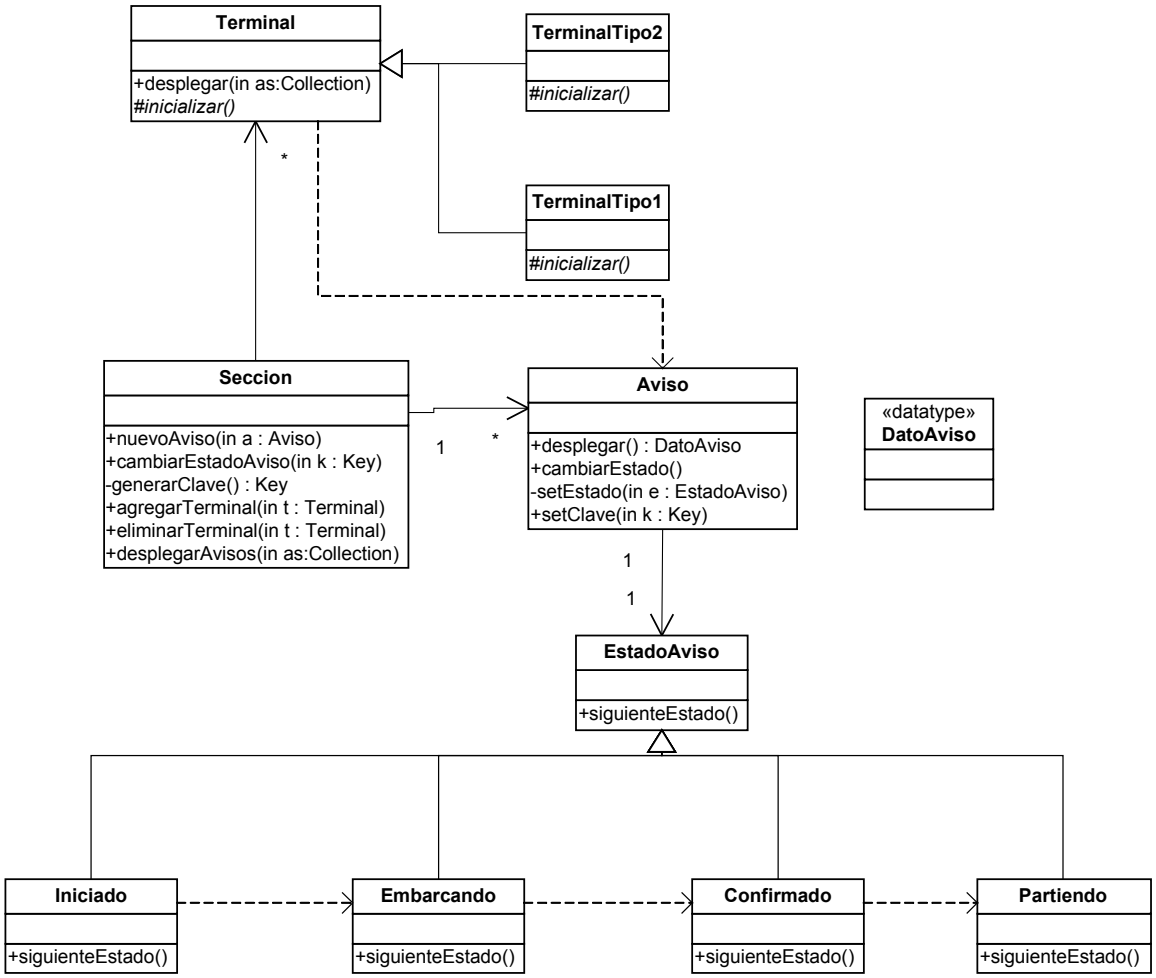




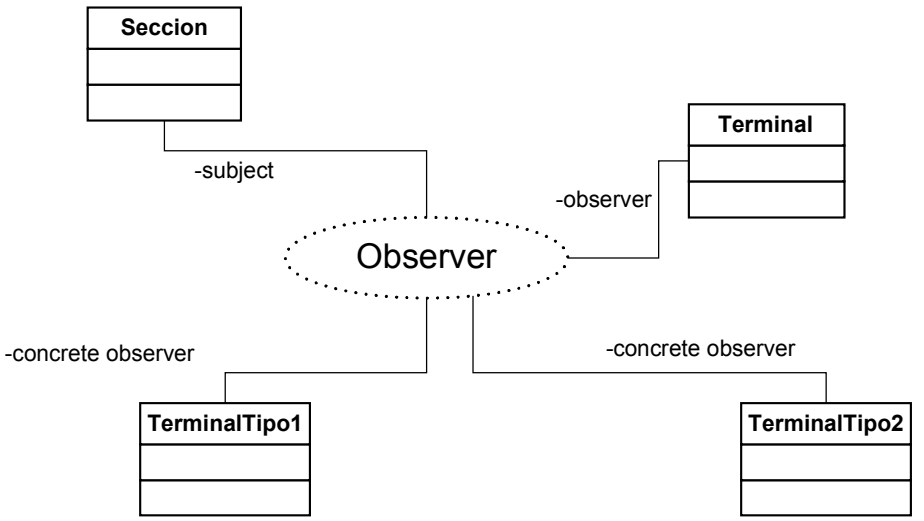


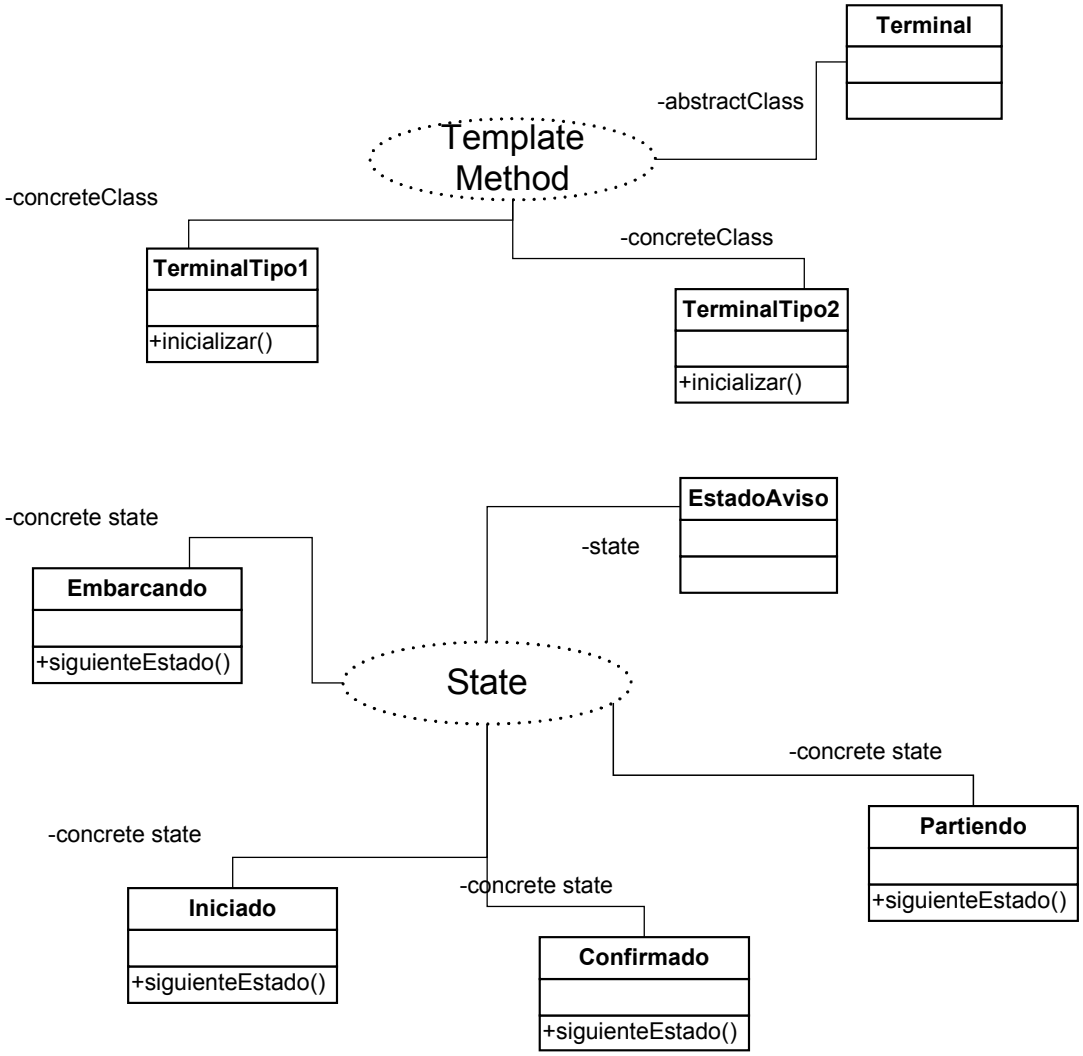

inicializar()
:TerminalTipo1
inicializar()
:TerminalTipo2
2: setEstado(e)
cambiarEstado()
:Aviso
1: e:=siguienteEstado()
:EstadoAviso
e:=siguienteEstado()
:Iniciado
1: e:=create()
:Confirmado
e:=siguienteEstado()
:Embarcando
1: e:=create()
:Partiendo
e:=siguienteEstado()
:Confirmado
1: e:=create()
:Embarcando
e:=siguienteEstado()
:Partiendo
1: e:=create()
:Finalizando

ii.


Terminal
+desplegar(in as:Collection)
#inicializar()
TerminalTipo2
#inicializar()
TerminalTipo1
#inicializar()
*
Seccion
+nuevoAviso(in a : Aviso)
+cambiarEstadoAviso(in k : Key)
-generarClave() : Key
+agregarTerminal(in t : Terminal)
+eliminarTerminal(in t : Terminal)
+desplegarAvisos(in as:Collection)
1
*
Aviso
+desplegar() : DatoAviso
+cambiarEstado()
-setEstado(in e : EstadoAviso)
+setClave(in k : Key)
«datatype»
DatoAviso
1
1
EstadoAviso
+siguienteEstado()
Iniciado
+siguienteEstado()
Embarcando
+siguienteEstado()
Confirmado
+siguienteEstado()
Partiendo
+siguienteEstado()

iii.


Seccion
-subject
Terminal
-observer
Observer
-concrete observer
-concrete observer
TerminalTipo1
TerminalTipo2


Terminal
-abstractClass
Template Method
-concreteClass
-concreteClass
TerminalTipo1
+inicializar()
TerminalTipo2
+inicializar()
EstadoAviso
-concrete state
-state
Embarcando
+siguienteEstado()
State
-concrete state
Partiendo
+siguienteEstado()
-concrete state
-concrete state
Iniciado
+siguienteEstado()
Confirmado
+siguienteEstado()

Problema 4 (25 puntos)

a) Ver transparencias 42, 43 y 44 del juego "05 – conceptos basicos oo".

b)
i.

```
// ConexionAbierta.hh

class ConexionAbierta
{
private:
      bool libre;
      Connection* connection;
      static ConexionAbierta* pool[N];
      static int cantidad;
      ConexionAbierta();

public:
      static ConexionAbierta* getInstance();
      static void vaciarPool();
      void liberar();
      void enviarMensaje(String datos);
      ~ConexionAbierta();
};

// ConexionAbierta.cc

int ConexionAbierta::cantidad = 0;
ConexionAbierta* ConexionAbierta::pool[N];

ConexionAbierta::ConexionAbierta()
{
      this->connection = new Connection();
      this->connection->open();
}

ConexionAbierta* ConexionAbierta::getInstance()
{
      ConexionAbierta* result;

      // Se busca alguna conexion libre
      for (int i = 0; i < ConexionAbierta::cantidad; i++)
      {
            result = ConexionAbierta::pool[i];
            if (result->libre)
            {
                  result->libre = false;
                  return result;
            }
      }

      // Si no se encontraron conexiones libres se intenta crear otra
         en caso que el pool no se haya llenado
      if (ConexionAbierta::cantidad < N)
      {
            result=ConexionAbierta::pool[ConexionAbierta::cantidad++]
                  = new ConexionAbierta();
            result->libre = false;
```

```
            return result;
      }

      // No hay conexiones disponibles
      return NULL;
}

void ConexionAbierta::vaciarPool()
{
      for (int i = 0; i < ConexionAbierta::cantidad; i++)
            delete ConexionAbierta::pool[i];

      ConexionAbierta::cantidad = 0;
}

void ConexionAbierta::liberar()
{
      this->libre = true;
}

void ConexionAbierta::enviarMensaje(String datos)
{
      this->connection->sendMessage(datos);
}

ConexionAbierta::~ConexionAbierta()
{
      this->connection->close();
      delete this->connection;
}
```

ii.

Modificaciones a la clase ConexionAbierta:

- Se debe modificar el cabezal de la operación `enviarMensaje` agregándole el modificador **virtual** de forma que sus invocaciones se despachen dinámicamente.
- Se modifica el método de la operación `getInstance` de forma de instanciar en el pool objetos de tipo `ConexionAbiertaLog`; la línea modificada es:

```
result = ConexionAbierta::pool[ConexionAbierta::cantidad++] =
new ConexionAbiertaLog();
```

```
// ConexionAbiertaLog.hh

class ConexionAbiertaLog : public ConexionAbierta
{
public:
      void enviarMensaje(String datos);
};

// ConexionAbiertaLog.cc

void ConexionAbiertaLog::enviarMensaje(String datos)
{
      // Se reutiliza el metodo de la clase base para enviar el
         mensaje
      this->ConexionAbierta::enviarMensaje(datos);

      // Se registra el mensaje
      Utils::registrarMensaje(datos);
}
```