Programación 4

SOLUCIÓN EXAMEN FEBRERO 2009

Problema 1 (35 puntos)

a)

Ver material de teórico "Análisis – Comportamiento del Sistema".

b)

i.

nuevoPedido(idC : integer) : Set(integer)
Pre:
Existe un Cliente en el Sistema cuyo atributo idC coincide con idC.
Post:
El Sistema recuerda la instancia de Cliente identificada por idC.
El Sistema recuerda un identificador autogenerado de Pedido.
Se devuelve un Set de integer conteniendo los valores del atributo idProd de todas las instancias de Producto existentes en el Sistema.

agregarProducto(idProd : integer)
Pre:
Existe en el sistema un producto cuyo atributo idProd coincide con idProd.
Post:
El Sistema recuerda el valor idProd.

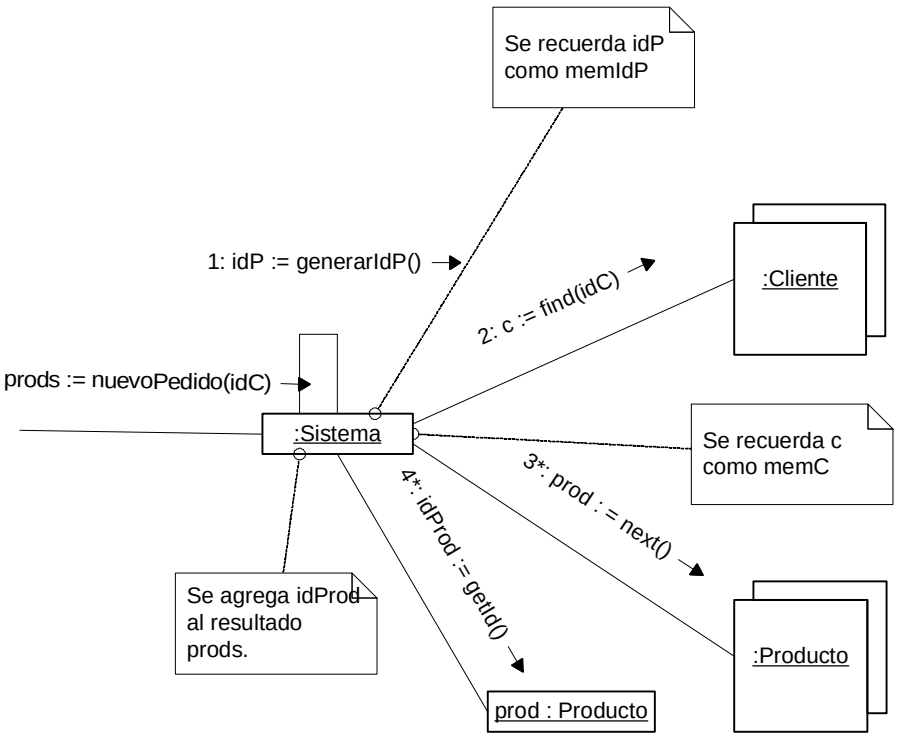
terminarPedido(confirmar : boolean)
Pre:
Existe una instancia de Cliente recordada por el Sistema.
Existen un identificador de Pedido y un conjunto no vacío de identificadores de Producto recordados por el Sistema.
Post:
Si confirmar = true, existe una nueva instancia (y se recuerda) de Pedido cuyo atributo idP coincide con el identificador de Pedido recordado por el Sistema. Se crea un link de la asociación realiza, entre el Cliente recordado por el Sistema y el nuevo Pedido. Se crea un link de la asociación constaDe, por cada identificador de Producto recordado por el Sistema, relacionando el respectivo Producto (cuyo atributo idProd coincide con el recordado) y el Pedido recordado por el Sistema.
Si confirmar = false, el Sistema deja de recordar el identificador de Pedido y los identificadores de Producto.

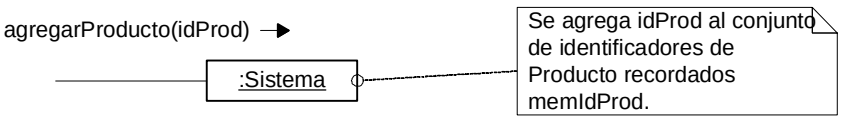
datosCredito(tarj : DTTarjeta , pagos : integer)
Pre:
Existe una instancia de Pedido recordada por el Sistema.
El valor del parámetros pagos es mayor que 0.
Post:
Existe una nueva instancia de Credito cuyos atributos tarjeta y cantPagos contienen los valores de tarj y pagos respectivamente.
Existe un link de la asociación sePagaCon, que relaciona la nueva instancia de Credito con el Pedido recordado por el Sistema.

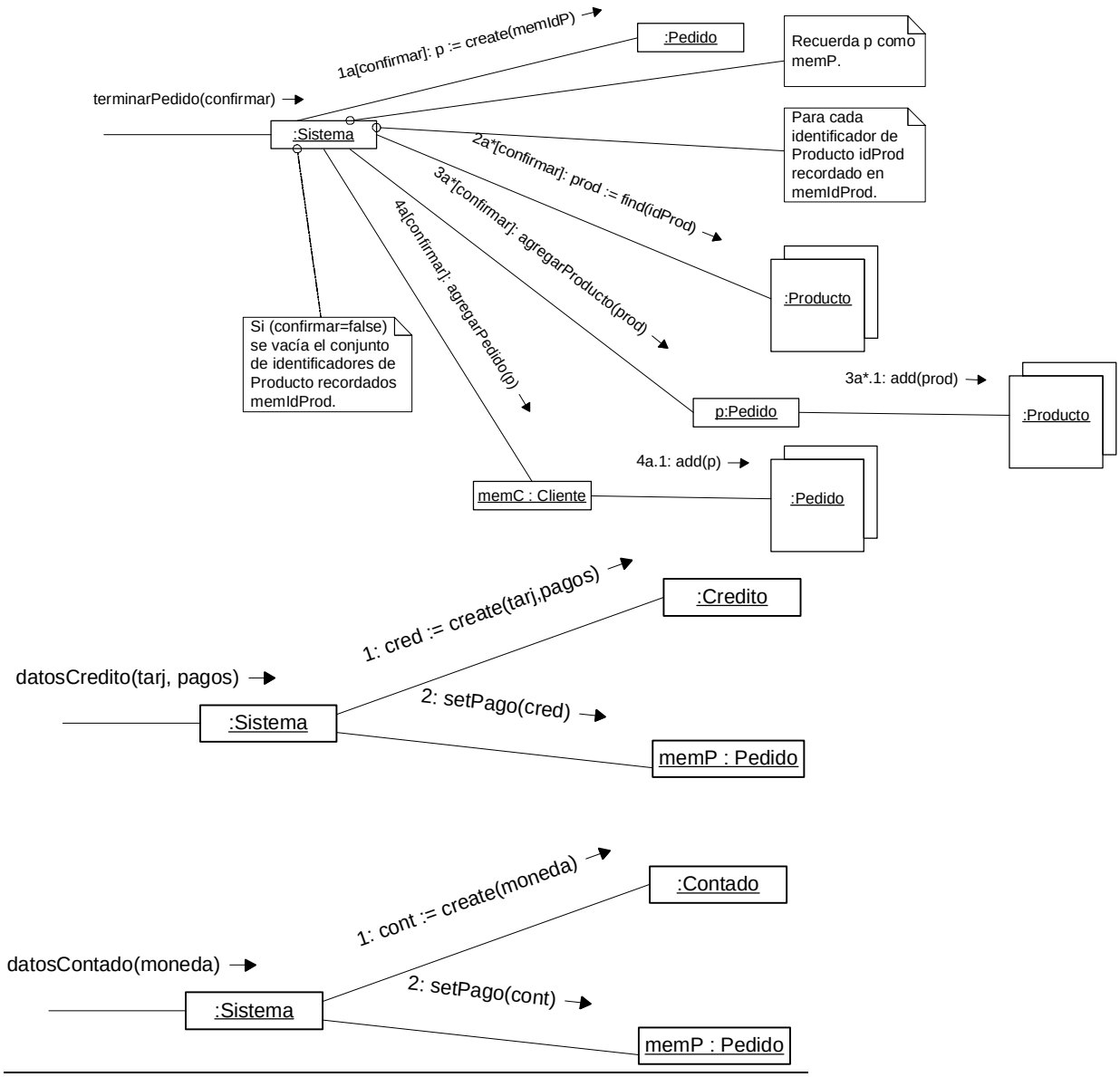
datosContado(moneda : DTMoneda)
Pre:
Existe una instancia de Pedido recordada por el Sistema.
Post:

Existe una nueva instancia de Contado cuyo atributo moneda contiene el valor de moneda.
Existe un link de la asociación sePagaCon, que relaciona la nueva instancia de Contado con el Pedido recordado por el Sistema.

ii.



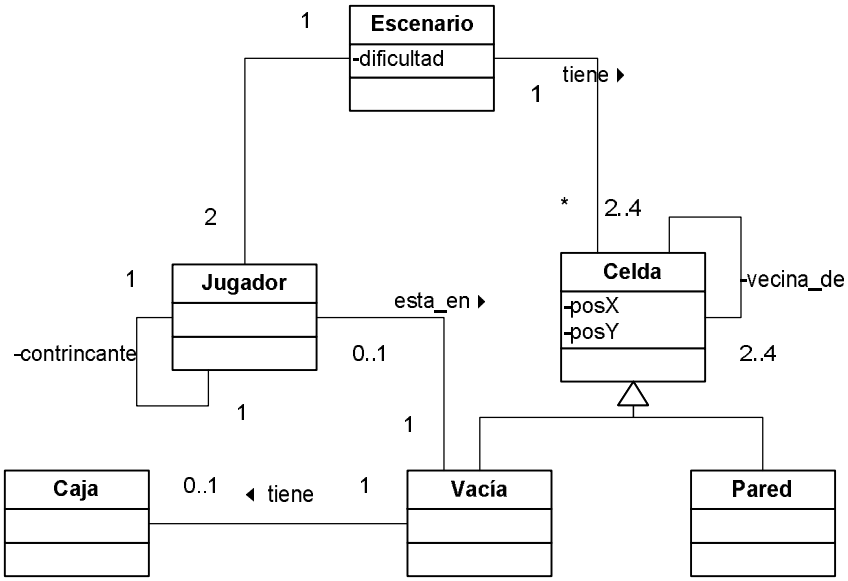



terminarPedido(confirmar)
:Sistema
1a[confirmar]: p := create(memIdP)
:Pedido
Recuerda p como memP.
Para cada identificador de Producto idProd recordado en memIdProd.
2a*[confirmar]: prod := find(idProd)
:Producto
3a*[confirmar]: agregarProducto(prod)
4a[confirmar]: agregarPedido(p)
Si (confirmar=false) se vacía el conjunto de identificadores de Producto recordados memIdProd.
3a*.1: add(prod)
p:Pedido
:Producto
4a.1: add(p)
memC : Cliente
:Pedido
datosCredito(tarj, pagos)
:Sistema
1: cred := create(tarj,pagos)
:Credito
2: setPago(cred)
memP : Pedido
datosContado(moneda)
:Sistema
1: cont := create(moneda)
:Contado
2: setPago(cont)
memP : Pedido

Problema 2 (30 puntos)

a) Ver teórico (lista de categorías de conceptos e identificación de sustantivos).

b)



Restricciones:

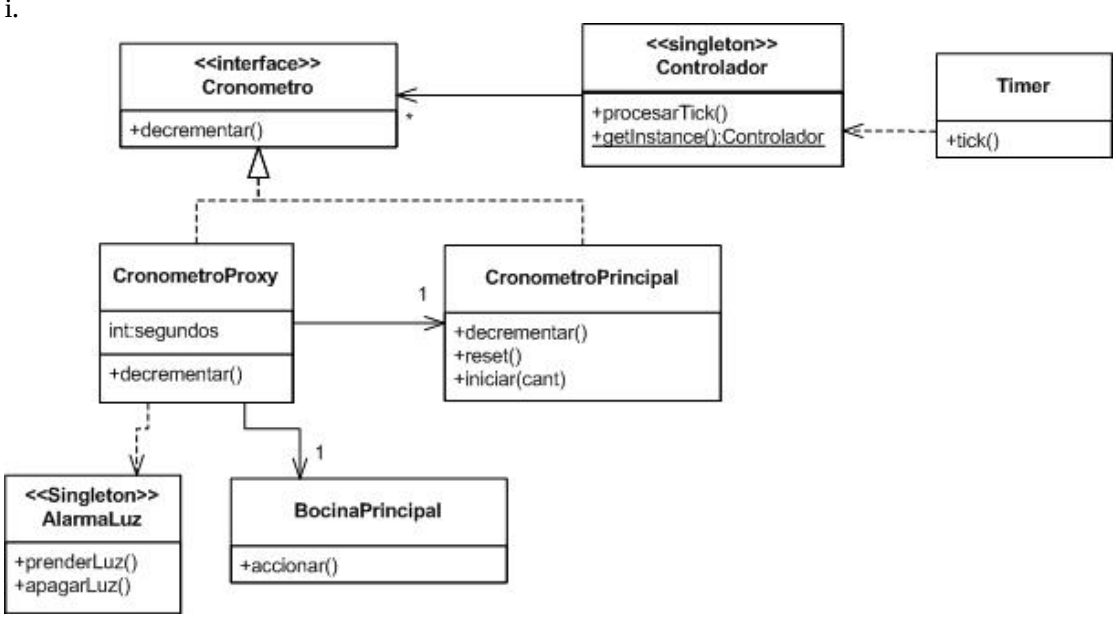
```
--Un jugador no puede jugar consigo mismo.
context Jugador inv:
   self.contrincante <> self

--Una celda no puede ser vecina consigo misma.
context Celda inv:
   self.vecinas->includes(self) = false

--Los escenarios de ambos jugadores debe ser el mismo.
context Jugador inv:
   self.escenario = self.contrincante.escenario

--El jugador debe estar en una celda vacía sin caja
context Jugador inv:
   self.vacia.caja->size() = 0

--El jugador debe estar en una celda del escenario en donde juega
context Jugador inv:
   self.escenario.celda->includes(self.vacia)
```

Existen además otras restricciones como:

- No hay 2 celdas con las mismas coordenadas.
- Celdas adyacentes deben tener coordenadas correspondientes a su ubicación.

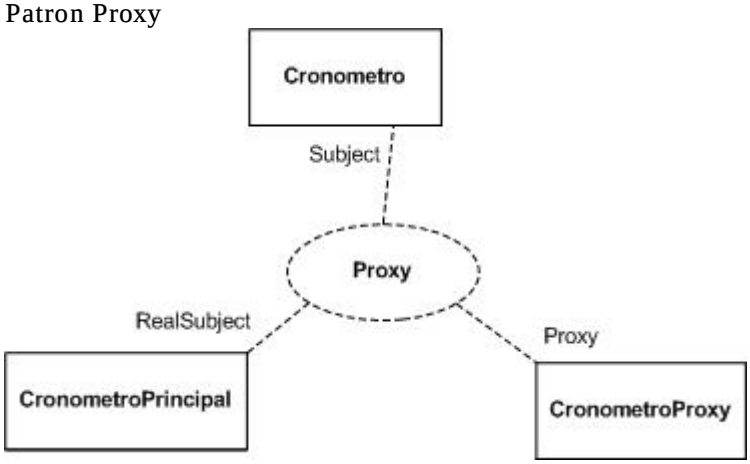
Problema 3 (35 puntos)

i.



ii.

Patron Proxy



iii.

----------Controlador.h

```
class Controlador {
public:
    Controlador();
    ~Controlador();
    void procesarTick();
    void agregarCronometro(Cronometro * crono);
private:
    List * cronos;
};
```

----------Controlador.cpp

```
Controlador::Controlador() {
    cronos = new LinkedList();
}

Controlador::agregarCronometro(Cronometro * crono){
    cronos->add(crono);
}

Controlador::procesarTick(){
    Iterator * it = cronos->getIterator();
    Cronometro * crono;
    while(it->hasNext()){
        crono = (Cronometro*)it->next();
        crono->decrementar();
    }
    delete it;
}

Controlador::~Controlador() {
      Iterator * it = cronos->getIterator();
      while(it->hasNext())
            it->removeCurrent();
      delete it;
      delete cronos;
}
```

----------Cronometro.h

```
class Cronometro{
public:
    Cronometro();
    virtual ~Cronometro();
    virtual void decrementar() = 0;
};
```

----------Cronometro.cpp

```
Cronometro::Cronometro(){}
Cronometro::~Cronometro(){}
```

----------CronometroProxy.h

```
#define TIEMPO = 600

class CronometroProxy : public Cronometro{
public:
    CronometroProxy(CronometroPrincipal * cronoPrinc);
    virtual ~CronometroProxy();
    virtual void decrementar();
private:
    int segundos;
    CronometroPrincipal * crono;
};
```

----------CronometroProxy.cpp

```
CronometroProxy::CronometroProxy(CronometroPrincipal * cronoPrinc) {
    crono = cronoPrinc;
    crono->iniciar(TIEMPO);
    segundos = TIEMPO;
    bocina = new BocinaPrincipal();
}

CronometroProxy::decrementar(){
    segundos--;
    crono->decrementar();
    if (segundos == 0){
        bocina->accionar();
        AlarmaLuz::getInstance()->prenderLuz();
        crono->reset();
        segundos = TIEMPO;
    } else if (segundos == TIEMPO - 1){
        AlarmaLuz::getInstance()->apagarLuz();
    }
}

CronometroProxy::~CronometroProxy() {
    delete bocina;
}
```