Programación 4

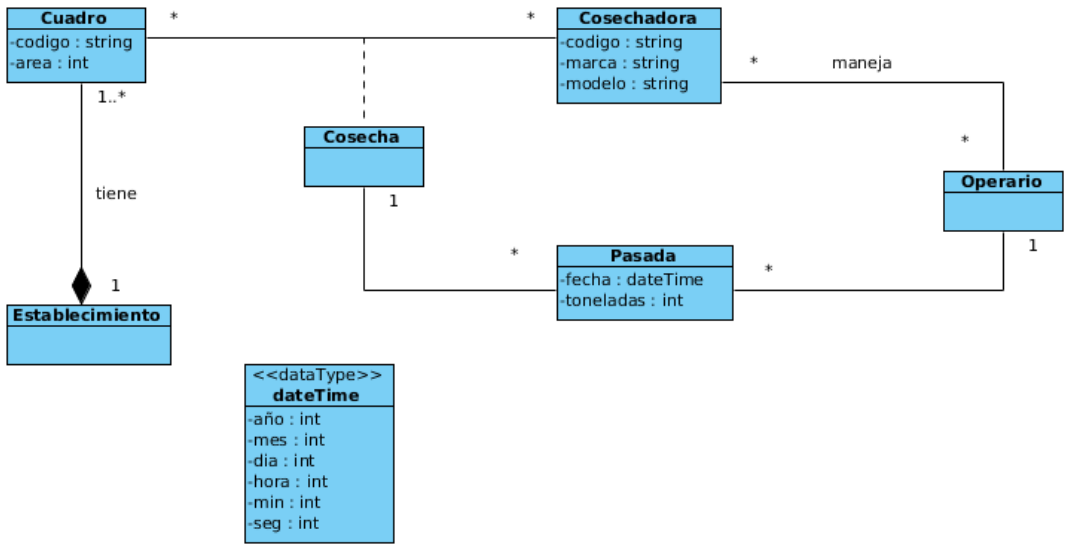
PARCIAL FINAL EDICIÓN 2013 - SOLUCIÓN

Problema 1 (30 puntos)

a) Ver teórico "06 – Análisis: Comportamiento del sistema", diapositivas: 48, 49 y 50.

Hablando en general, las pre y post condiciones de un contrato de software de una operación se especifican o refieren a condiciones sobre el estado del sistema antes (pre) y el estado del sistema después (post) de la invocación a la operación. Las precondiciones refieren además a los argumentos de la operación (si tiene). Las postcondiciones refieren además al valor retornado por la operación (si tiene).

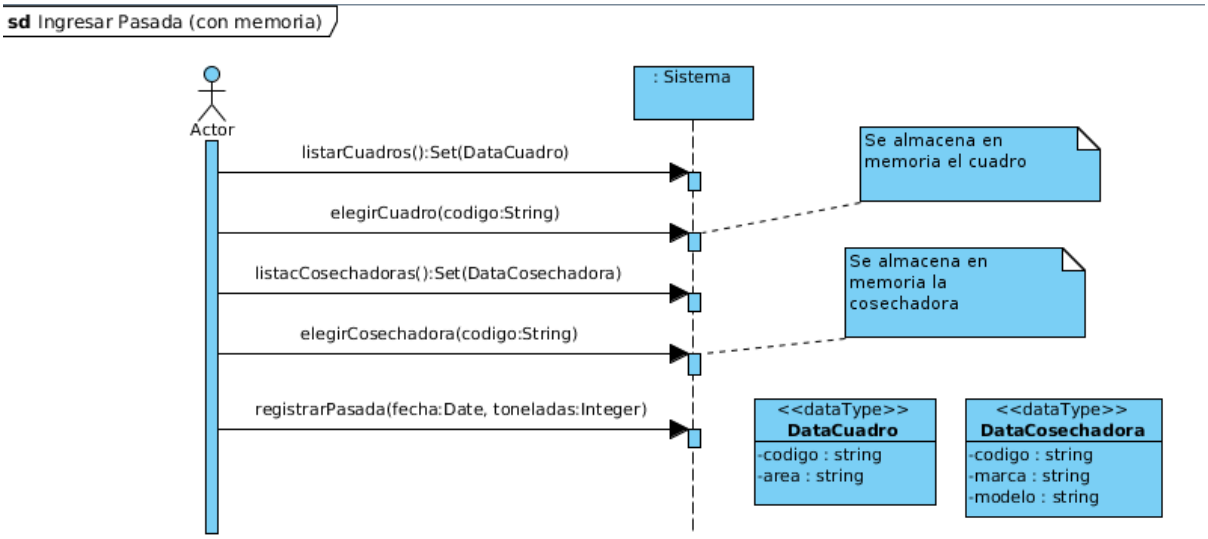
b)



Restricciones no estructurales:

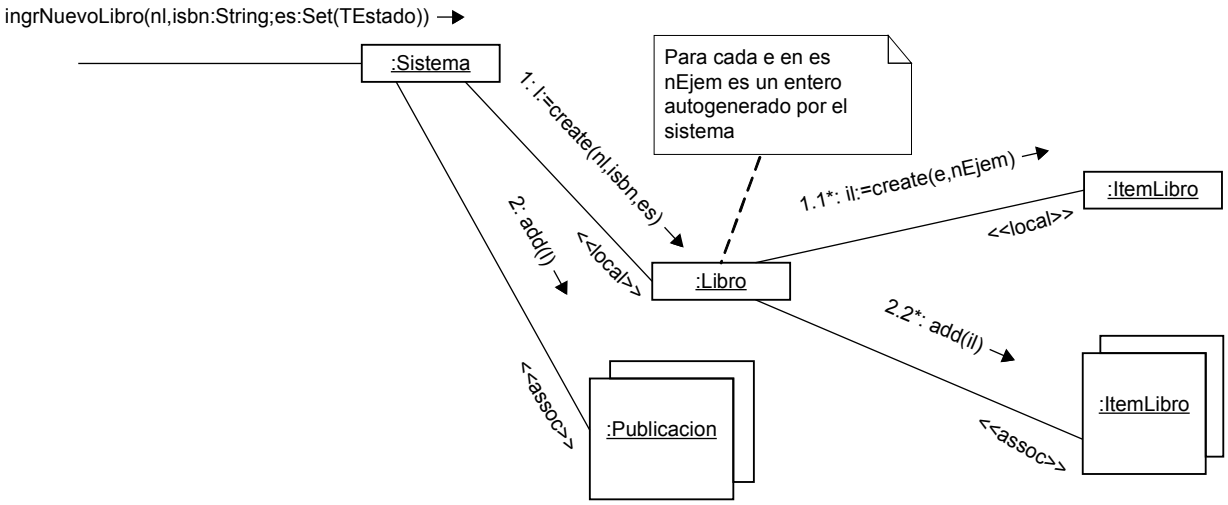
- No existen dos "Pasada" con la misma fecha para el par Cuadro-Cosechadora
- No existen dos cosechadoras con el mismo código
- No existen dos cuadros con el mismo codigo
- La "Pasada" registrada a un operario fue cosechada por una cosechadora que maneja el operario.

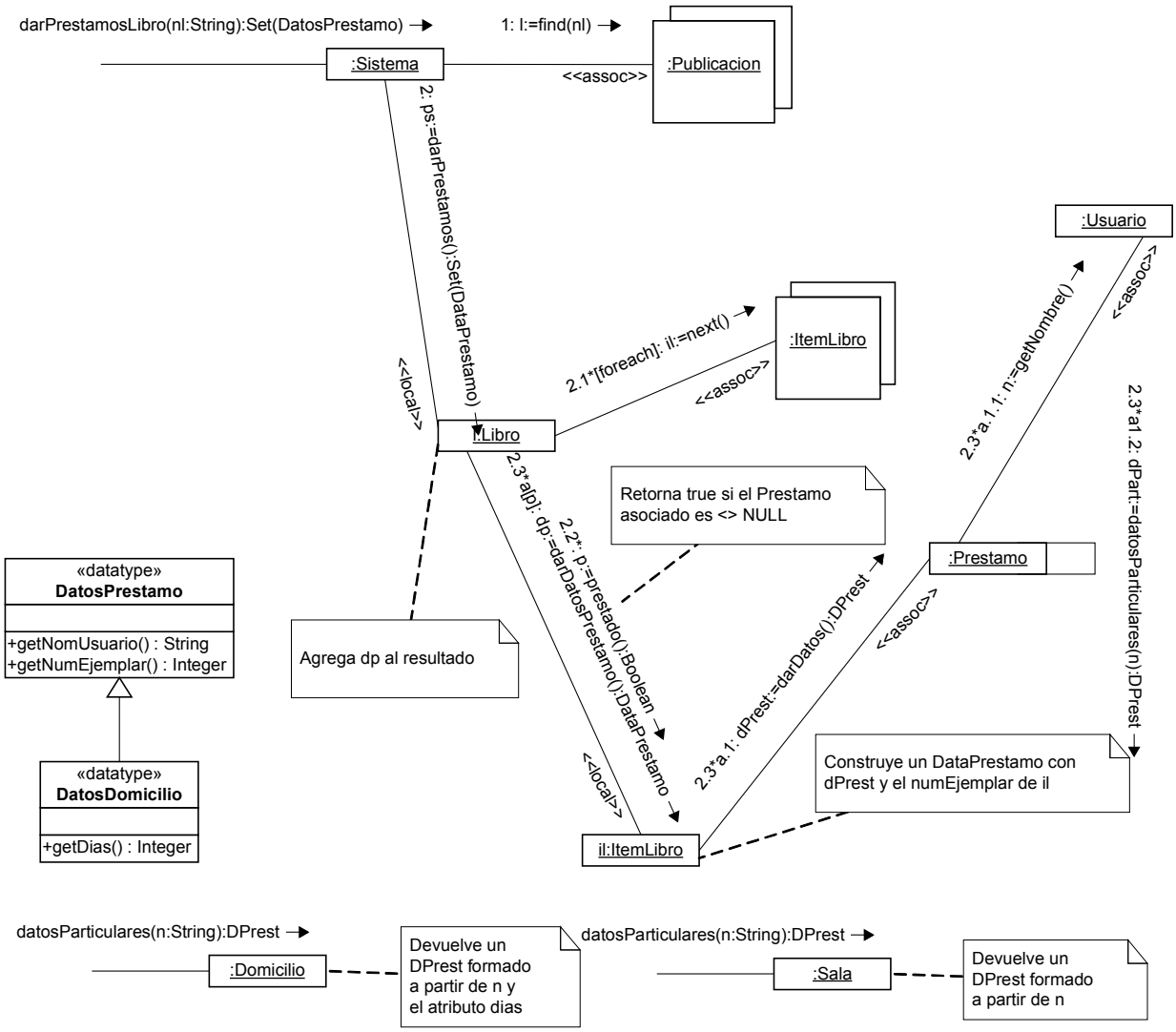
c)


sd Ingresar Pasada (con memoria)
Actor
: Sistema
listarCuadros():Set(DataCuadro)
elegirCuadro(codigo:String)
listacCosechadoras():Set(DataCosechadora)
elegirCosechadora(codigo:String)
registrarPasada(fecha:Date, toneladas:Integer)
Se almacena en memoria el cuadro
Se almacena en memoria la cosechadora
<<dataType>>
DataCuadro
-codigo : string
-area : string
<<dataType>>
DataCosechadora
-codigo : string
-marca : string
-modelo : string

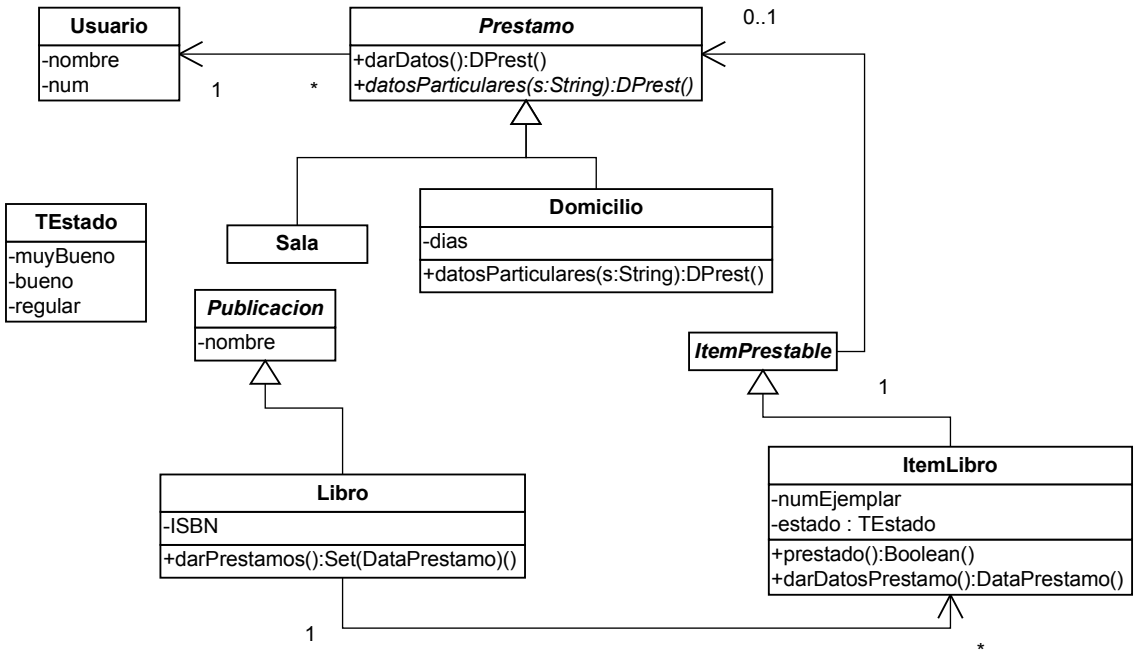
Problema 2 (35 puntos)

i.

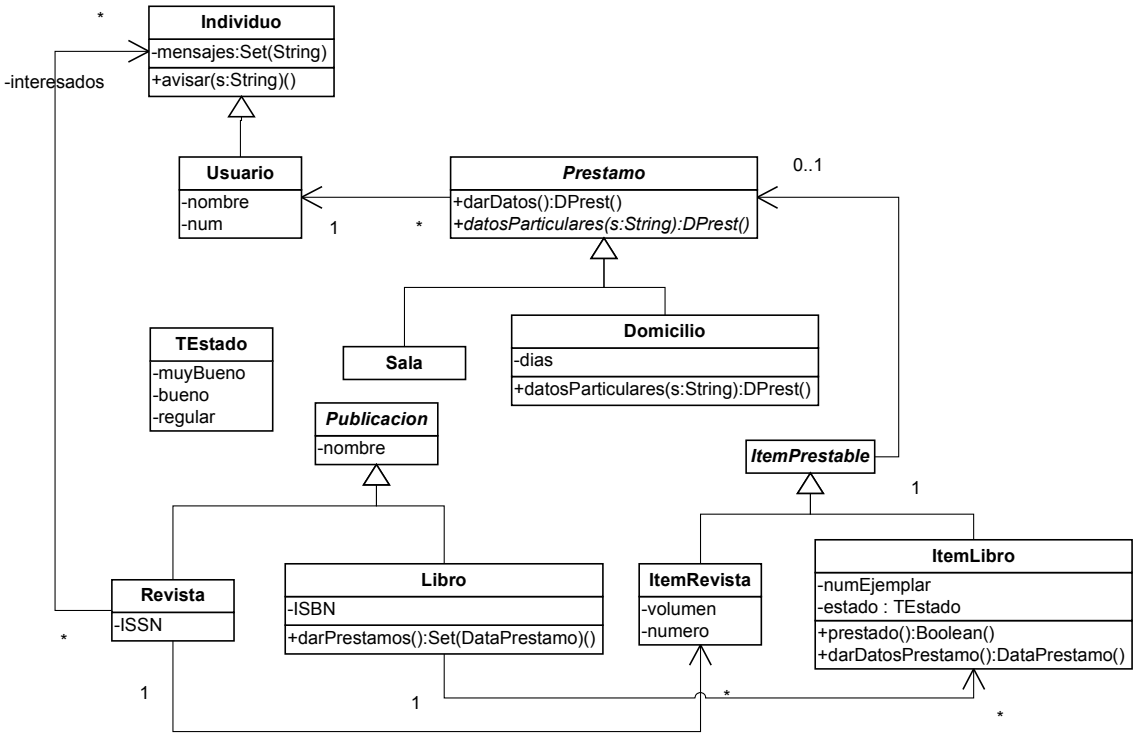



darPrestamosLibro(nl:String):Set(DatosPrestamo)
1: l:=find(nl)
:Sistema
<<assoc>>
:Publicacion
2: ps:=darPrestamos():Set(DataPrestamo)
<<local>>
2.1*[foreach]: il:=next()
:ItemLibro
<<assoc>>
l:Libro
2.3*a[p]: dp:=darDatosPrestamo():DataPrestamo
2.2*: p:=prestado():Boolean
Retorna true si el Prestamo asociado es <> NULL
:Usuario
<<assoc>>
2.3*a.1.1: n:=getNombre()
2.3*a1.2: dPart:=datosParticulares(n):DPrest
:Prestamo
<<assoc>>
2.3*a.1: dPrest:=darDatos():DPrest
«datatype»
DatosPrestamo
+getNomUsuario() : String
+getNumEjemplar() : Integer
Agrega dp al resultado
«datatype»
DatosDomicilio
+getDias() : Integer
<<local>>
il:ItemLibro
Construye un DataPrestamo con dPrest y el numEjemplar de il
datosParticulares(n:String):DPrest
:Domicilio
Devuelve un DPrest formado a partir de n y el atributo dias
datosParticulares(n:String):DPrest
:Sala
Devuelve un DPrest formado a partir de n

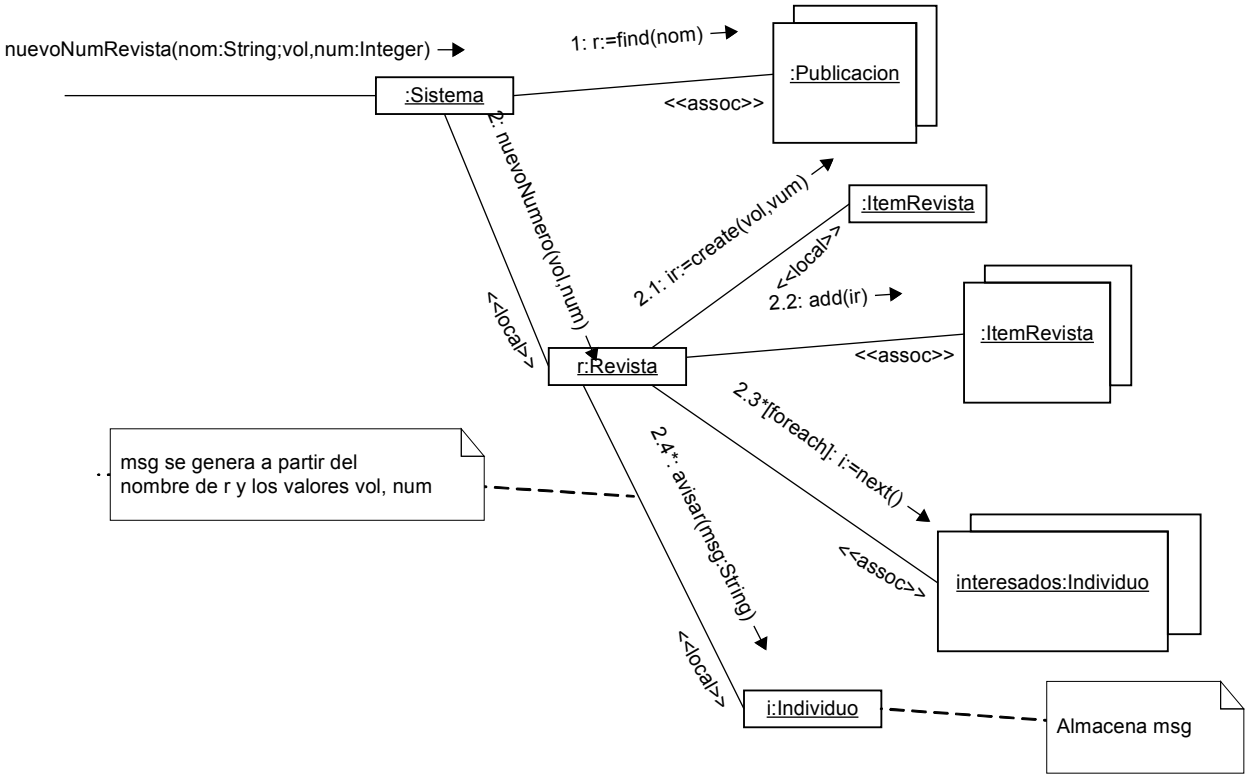
ii.


Usuario
-nombre
-num
1
*
Prestamo
+darDatos():DPrest()
+datosParticulares(s:String):DPrest()
0..1
TEstado
-muyBueno
-bueno
-regular
Sala
Domicilio
-dias
+datosParticulares(s:String):DPrest()
Publicacion
-nombre
ItemPrestable
1
Libro
-ISBN
+darPrestamos():Set(DataPrestamo)()
ItemLibro
-numEjemplar
-estado : TEstado
+prestado():Boolean()
+darDatosPrestamo():DataPrestamo()
1
*

iii.


Individuo
-mensajes:Set(String)
+avisar(s:String)()
-interesados
Usuario
-nombre
-num
Prestamo
+darDatos():DPrest()
+datosParticulares(s:String):DPrest()
0..1
TEstado
-muyBueno
-bueno
-regular
Sala
Domicilio
-dias
+datosParticulares(s:String):DPrest()
Publicacion
-nombre
ItemPrestable
Revista
-ISSN
Libro
-ISBN
+darPrestamos():Set(DataPrestamo)()
ItemRevista
-volumen
-numero
ItemLibro
-numEjemplar
-estado : TEstado
+prestado():Boolean()
+darDatosPrestamo():DataPrestamo()

iv.


nuevoNumRevista(nom:String;vol,num:Integer)
:Sistema
1: r:=find(nom)
<<assoc>>
:Publicacion
2: nuevoNumero(vol,num)
<<local>>
2.1: ir:=create(vol,vum)
:ItemRevista
<<local>>
2.2: add(ir)
:ItemRevista
<<assoc>>
r:Revista
2.3*[foreach]: i:=next()
<<assoc>>
interesados:Individuo
2.4*: avisar(msg:String)
<<local>>
i:Individuo
msg se genera a partir del
nombre de r y los valores vol, num
Almacena msg

v.

Se usa el patrón Observer:

Subject: Revista

Observer: Individuo

Concrete observer: Usuario

Problema 3 (35 puntos)

Parte i:

```
class Subject {
public:
   void notifyObservers(Event *ev);
   void addObserver(Observer *obs);
   void removeObserver(Observer *obs);
   void clearObservers();
   virtual ~Subject() {};

   virtual void onCancel(Observer *obs) = 0;
private:
   set<Observer *> observers;
};

void Subject::notifyObservers(Event *ev){
   set<Observer *>::iterator it;
   for(it = observers.begin(); it != observers.end(); ++it){
      (*it)->notify(ev);
      if(ev->isCancelled()){
         onCancel(*it);
         break;
      }
   }
}

void Subject::addObserver(Observer *obs){
   observers.insert(obs);
}

void Subject::removeObserver(Observer *obs){
   observers.erase(obs);
}

void Subject::clearObservers(){
   observers.clear();
}

class Observer {
public:
   virtual void notify(Event *ev) = 0;
   virtual ~Observer() {}
};
```

Parte ii:

```
class Sistema: public Subject {
public:
   string comprar(int idProd, int cant);
   void onCancel(Observer *tienda);
private:
   Tienda *tiendaConStock;
};

string Sistema::comprar(int idProd, int cant){
   tiendaConStock = NULL;
   EventoCompra *c = new EventoCompra(idProd, cant);
   notifyObservers(c);
   if(tiendaConStock != NULL)
      return tiendaConStock->getNombre();
   else
      throw invalid_argument("No hay stock");
   delete c;
}

void Sistema::onCancel(Observer *tienda){
   tiendaConStock = dynamic_cast<Tienda *>(tienda);
}

class Tienda: public Observer {
public:
   string getNombre();
   void notify(Event *ev);
private:
   map<int, Producto *> productos;
   string nombre;
};

void Tienda::notify(Event* ev){
   EventoCompra *c = dynamic_cast<EventoCompra *>(ev);
   Producto *p = productos[c->getIdProd()];
   if(p != NULL && p->getStock() >= c->getCant()){
      int nuevoStock = p->getStock() - c->getCant();
      p->setStock(nuevoStock);
      ev->cancel();
   }
}
```