Programación 4

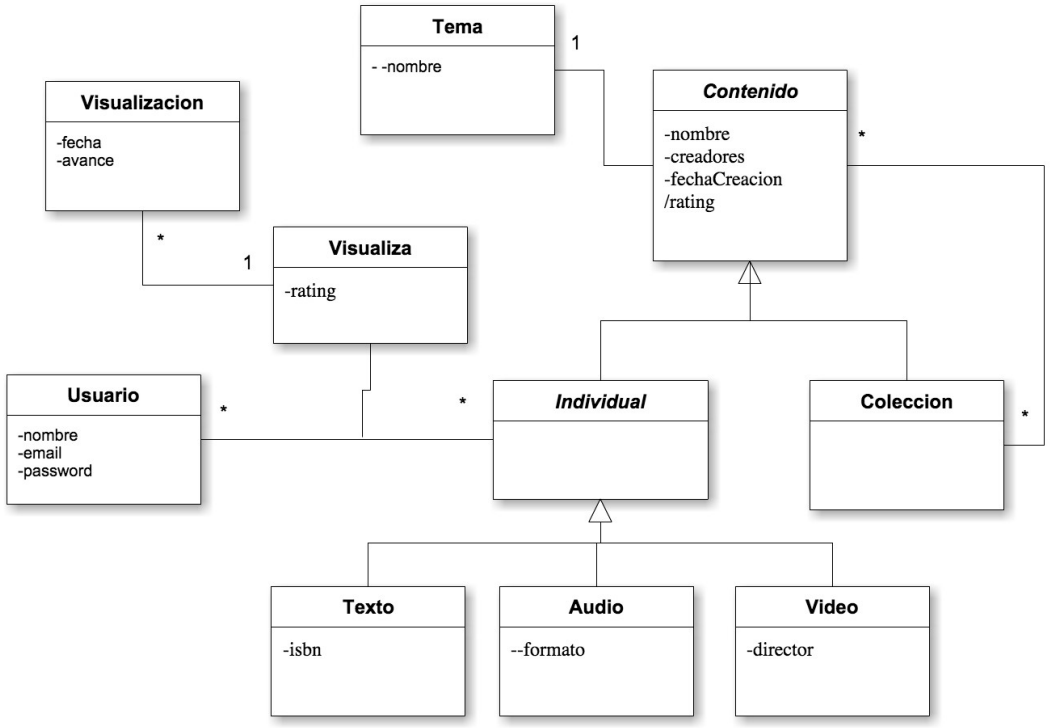
PARCIAL FINAL EDICIÓN 2014 – SOLUCIÓN

Problema 1

Parte 1.a

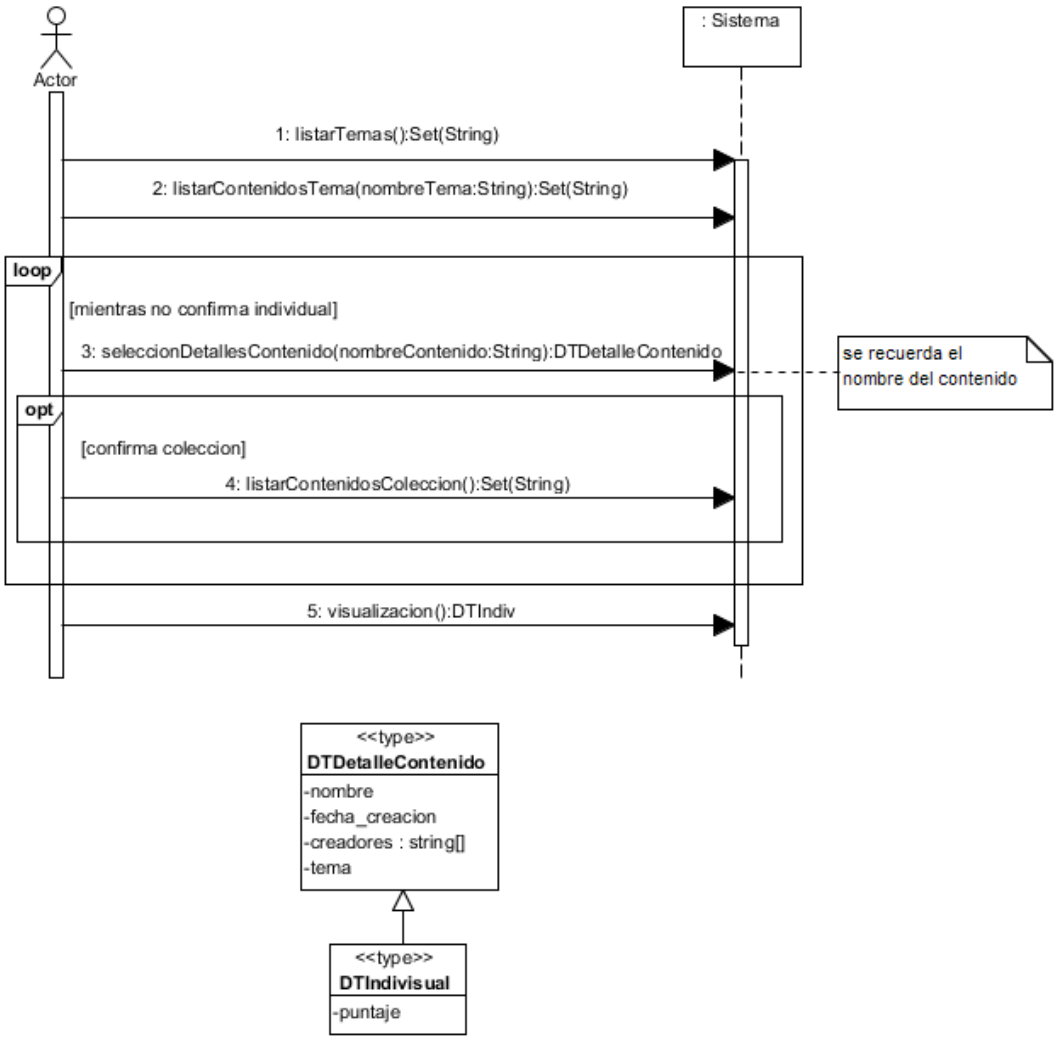
Ver diapositivas clase 03 Conceptos Básicos de Orientación a Objetos 1era Parte

Parte 1.b



Restricciones

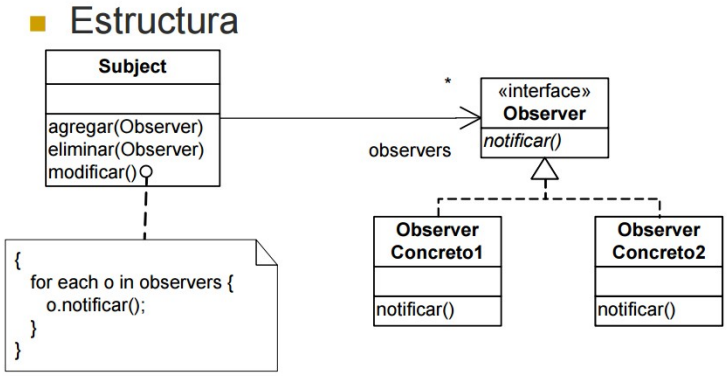
-Unicidad de Usuario.nombre, Contenido.nombre, Tema.nombre.
-Unicidad de isbn (opcional)
-No existen colecciones que se contengan a sí mismas (recursivamente).
-Ninguna fecha de visualización es anterior a la fecha de creación del contenido asociado.
-Los valores de rating son enteros en [0..5]
- Los valores de avance son enteros en [0, 100]
-El rating de un contenido se determina en base a todas las instancias de Visualiza.rating del o los contenidos individuales asociados.


Actor
: Sistema
1: listarTemas():Set(String)
2: listarContenidosTema(nombreTema:String):Set(String)
loop
[mientras no confirma individual]
3: seleccionDetallesContenido(nombreContenido:String):DTDetalleContenido
se recuerda el nombre del contenido
opt
[confirma coleccion]
4: listarContenidosColeccion():Set(String)
5: visualizacion():DTIndiv

<<type>>
DTDetalleContenido
-nombre
-fecha_creacion
-creadores : string[]
-tema
<<type>>
DTIndivisual
-puntaje

Problema 2

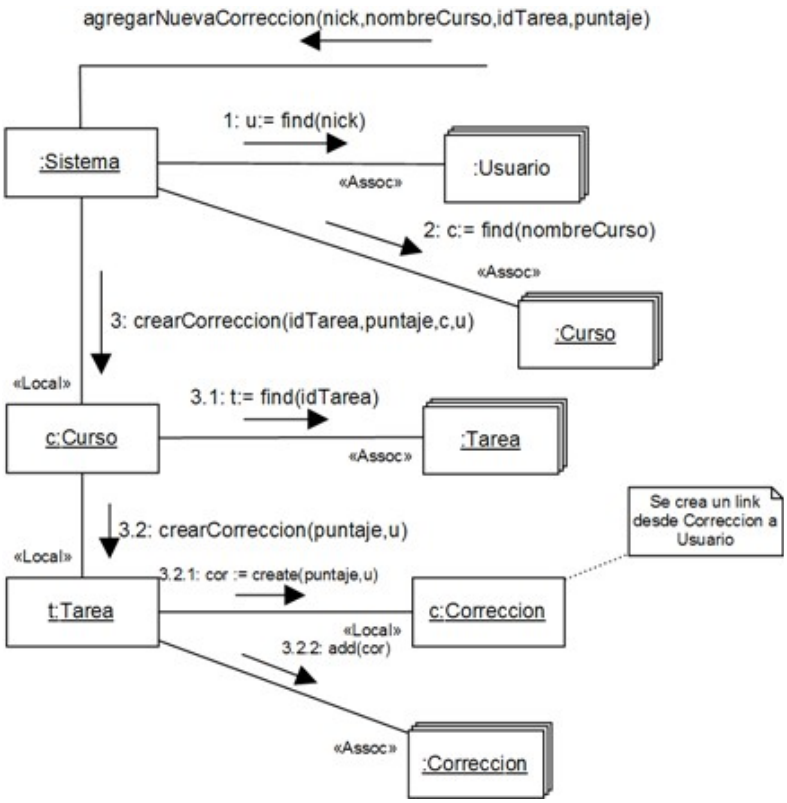
Parte a)
Problema tipo:
Definir una dependencia 1-n entre objetos, de forma que cuando uno cambie de estado todos los dependientes sean notificados. (Diapositiva 51 teórico / 2015)
Estructura:



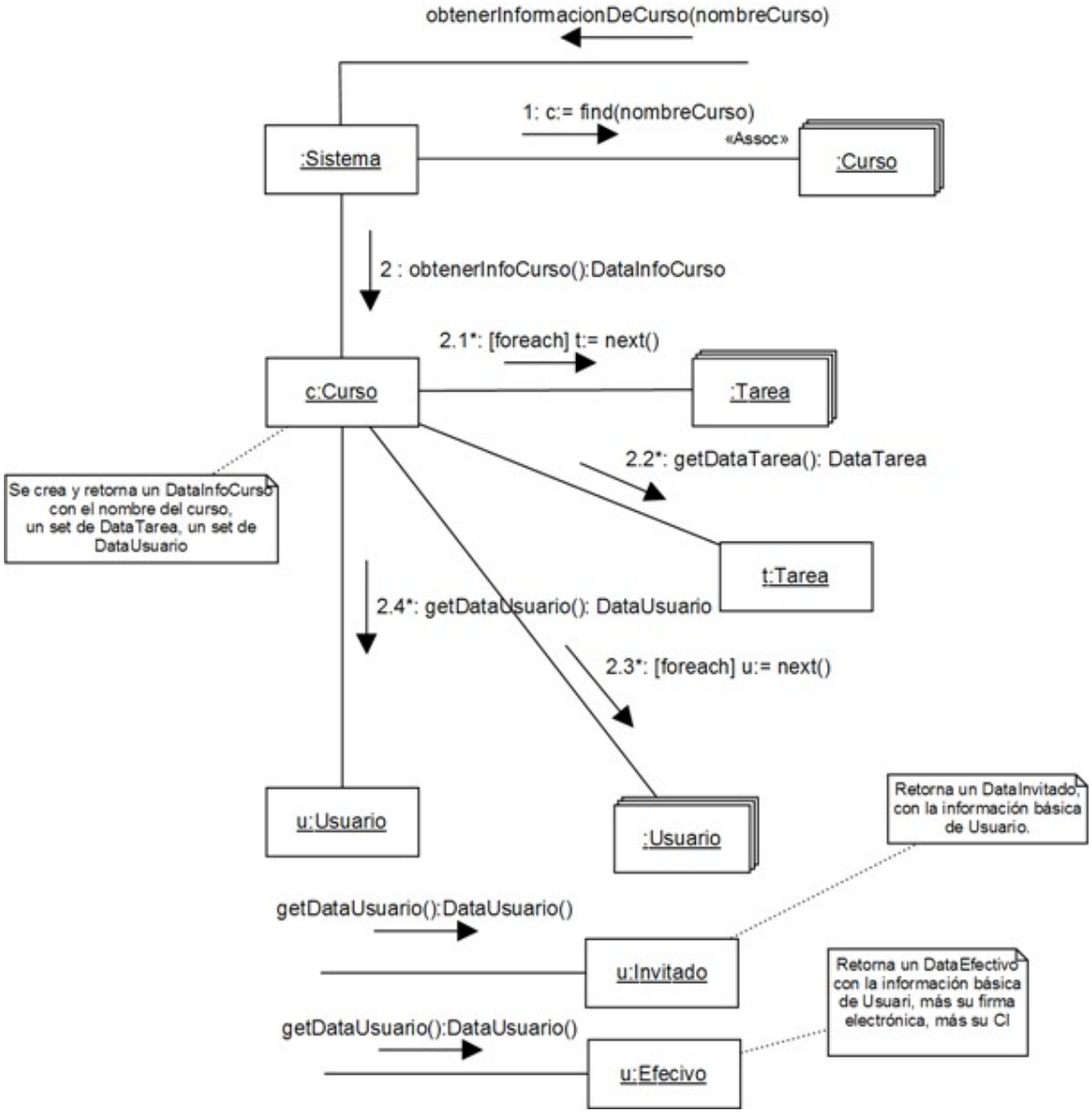
Participantes:
Subject, Observer y Observer Concreto.

Parte b)

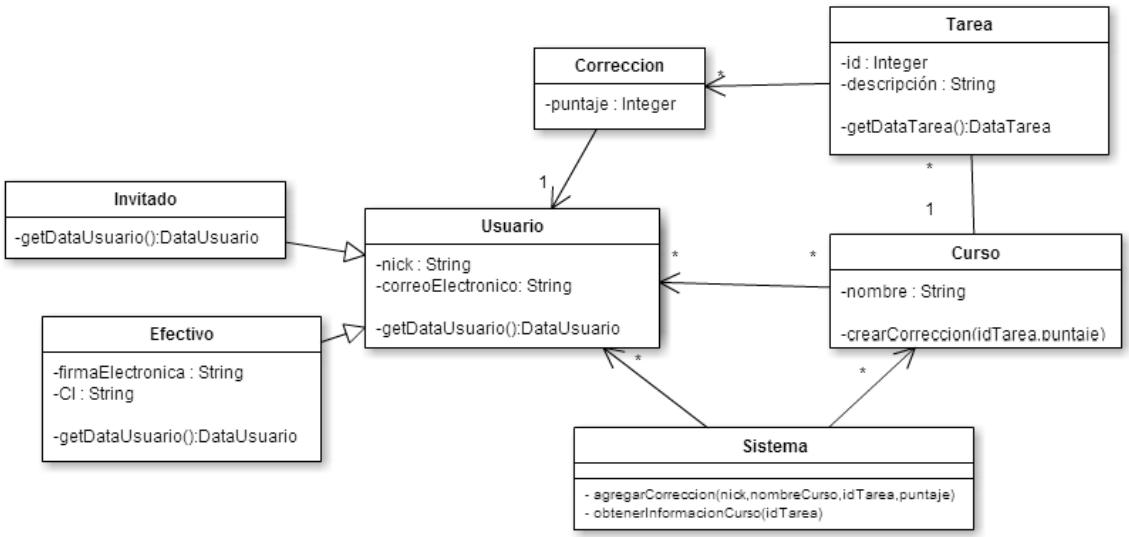
i. AgregarNuevaCorrección



obtenerInformaciónDeCurso



ii. Diagrama de clases de diseño



Problema 3

```
//Cliente.h
class Cliente {
public:
    Cliente();
    virtual ~Cliente();
private:
    string ci;
    string nombre;
    string tel;
    string dir;
    Posicion posicion;
};

//Cliente.cpp
Cliente::Cliente() {
}

Cliente::~Cliente() {
}

*************************
//Producto.h
class Producto {  // Opcion 1
//class Producto: public ICollectible { // Opcion 2
public:
    Producto();
    virtual ~Producto();
private:
    string nombre;
    string descr;
    float precio;
    int cantidad;
};

//Producto.cpp
Producto::Producto() {
}

Producto::~Producto() {
}

*************************
//Pago.h
class Pago {
public:
    Pago(float monto);
    virtual ~Pago();
private:
    float monto;
};

//Pago.cpp
Pago::Pago(float monto) {
    this->monto = monto;
}

Pago::~Pago() {
}
```

```
*************************
//Contado.h
class Contado: public Pago {
public:
    Contado(float monto, float descuento);
    virtual ~Contado();
private:
    float descuento;
};

//Contado.cpp
Contado::Contado(float monto, float descuento): Pago(monto) {
    this->descuento = descuento;
}

Contado::~Contado() {
}

*************************
//Tarjeta.h
class Tarjeta: public Pago {
public:
    Tarjeta(float monto, TTarjeta tarjeta);
    virtual ~Tarjeta();
private:
    TTarjeta tarjeta;
};

//Tarjeta.cpp
Tarjeta::Tarjeta(float monto, TTarjeta tarjeta): Pago(monto) {
    this->tarjeta = tarjeta;
}

Tarjeta::~Tarjeta() {
}

*************************
//TTarjeta.h
enum TTarjeta{
    Viza,
    MastroCar,
    Oka
};

*************************
//ServicioMapa.h
class ServicioMapa {
public:
    static ServicioMapa* getInstance();
    list<string> obtenerRuta(Posicion p);
    virtual ~ServicioMapa();
private:
    static ServicioMapa *instance;
    ServicioMapa();
};

//ServicioMapa.cpp
ServicioMapa *ServicioMapa::instance = NULL;

ServicioMapa::ServicioMapa(){}
```

```
ServicioMapa* ServicioMapa::getInstance() {
    if (instance==NULL)
        instance = new ServicioMapa();
    return instance;
}

list<string> ServicioMapa::obtenerRuta(Posicion p) {
      // NO SE IMPLEMENTA EL METODO
}

ServicioMapa::~ServicioMapa() {
}

*************************
//OrdenCompra.h

class OrdenCompra {
public:
    OrdenCompra();
    OrdenCompra(Cliente *cliente, Producto *producto);
    void agregarProducto(Producto *producto);
    void calcularTotal();
    void generarPago(DataPago *pago);
    void setRutaDespacho();
    virtual ~OrdenCompra();
private:
    float total;
    list<string> ruta;
    set<Producto*> productos; // Opcion 1
    //ICollection *productos; // Opcion 2
    Cliente *cliente;
    Pago *pago;
};

//OrdenCompra.cpp

OrdenCompra::OrdenCompra() {
}

OrdenCompra::OrdenCompra(Cliente *cliente, Producto* producto) {
// ---------------------------------------------
// Opcion 1
    this->productos.insert(producto);
// Opcion 2
//this->productos = new List();
//this->productos->add(producto);
// ---------------------------------------------
    this->total = 0;
    this->cliente = cliente;
    this->pago = NULL;
}

void OrdenCompra::agregarProducto(Producto* producto) {
// Opcion 1
    this->productos.insert(producto);
// Opcion 2
//this->productos->add(producto);
}
```

```
void OrdenCompra::calcularTotal() {
    float parcial = 0;

    // ****** Opcion 1 ******* //
    set<Producto*>::iterator prod;
    for(prod=this->productos.begin(); prod!=this->productos.end();
prod++){
        Producto *producto = *prod;
        parcial += producto->getCantidad() * producto->getPrecio();
    }
    // ****** Opcion 2 ******* //
//    IIterator *it = this->productos->getIterator();
//    while (it->hasCurrent()) {
//        Producto *producto = (Producto *)it->getCurrent();
//            parcial += producto->getCantidad() * producto->getPre-
cio();
//            it->next();
//    }
    // ********************** //
    this->total = parcial;
}

void OrdenCompra::generarPago(DataPago *pago) {
    Pago *tpago;
    DataContado *contado;
    DataTarjeta *tarjeta;

    contado = dynamic_cast<DataContado*> (pago);
    if (contado != NULL){
        tpago = new Contado(contado->getMonto(), contado->getDescuen-
to());
    }else{
        tarjeta = dynamic_cast<DataTarjeta*> (pago);
        tpago = new Tarjeta(tarjeta->getMonto(), tarjeta-
>getTarjeta());
    }
    this->pago = tpago;
}

void OrdenCompra::setRutaDespacho() {
    ServicioMapa *svrm = ServicioMapa::getInstance();
    this->ruta = svrm->obtenerRuta(this->cliente->getPosicion());
}

OrdenCompra::~OrdenCompra() {
    // ****** Opcion 1 ******* //
    set<Producto*>::iterator prod;
    for(prod=this->productos.begin(); prod!=this->productos.end();
prod++){
        delete *prod;
    }
    // ****** Opcion 2 ******* //
//    IIterator *it = this->productos->getIterator();
//    while (it->hasCurrent()) {
//        Producto *producto = (Producto *)it->getCurrent();
//            delete producto;
//            it->next();
//    }
    // ********************** //
    delete this->pago;
```

```
}
```