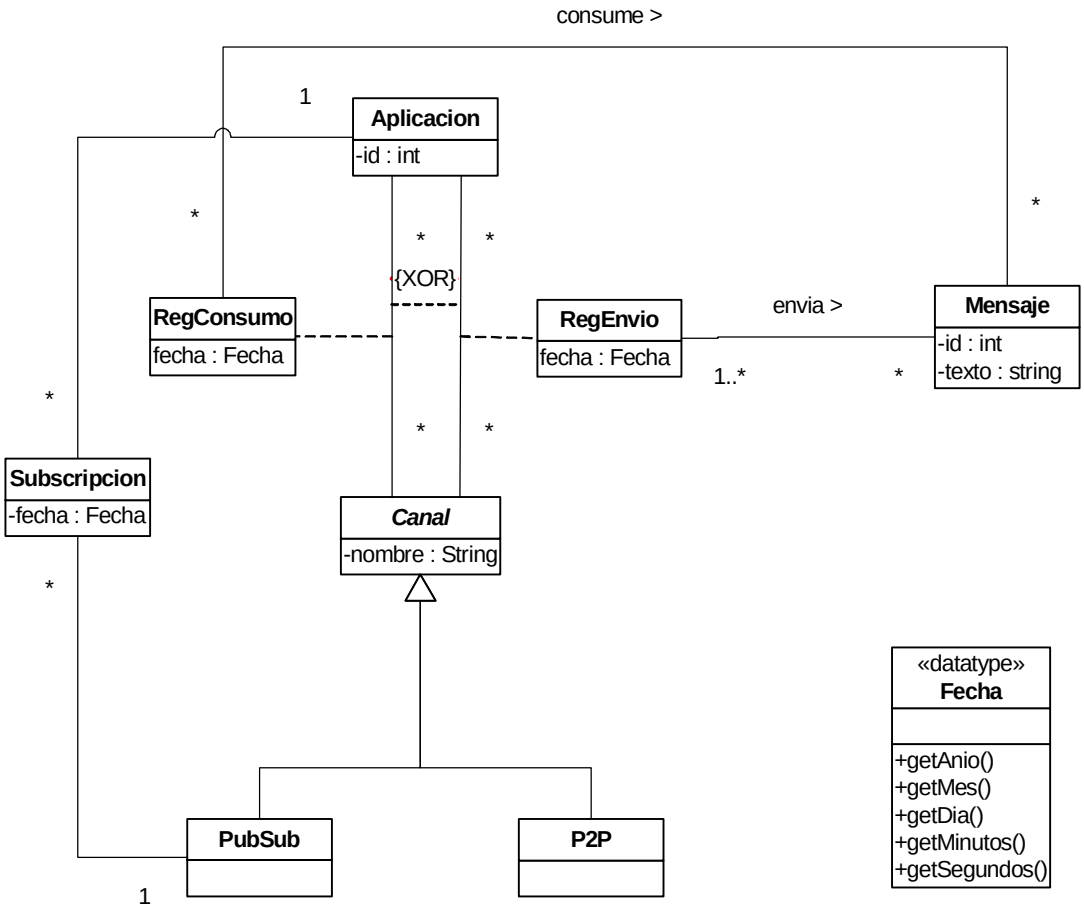
# Programación 4

## EXAMEN FEBRERO 2008

## SOLUCION

Problema 1 (25 puntos)

### Modelo




### Restricciones

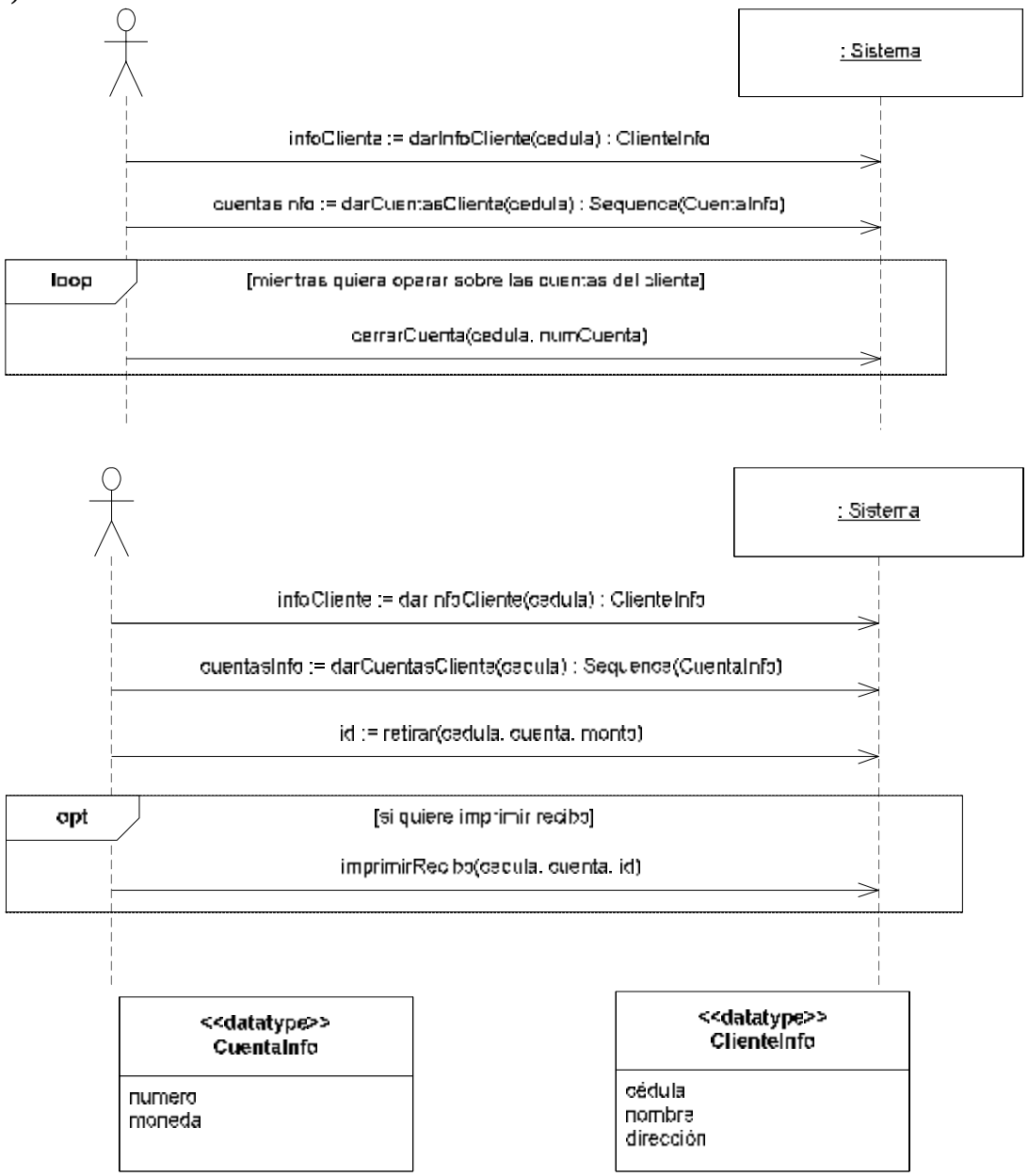
```
-- El identificador de la aplicación es único.
context Aplicacion inv:
    Aplicacion.allInstances()->isUnique(id)

-- El identificador del mensaje es único.
context Mensaje inv:
    Mensaje.allInstances()->isUnique(id)

-- El nombre del canal es único.
context Canal inv:
    Canal.allInstances()->isUnique(nombre)

-- Un mensaje consumido por una aplicación debe haber sido previamente enviado
por otra aplicación.
context Canal inv:
    self.reqConsumo.mensaje->forAll(m | self.reqEnvio.mensaje->includes(m))
```

```
-- Un mensaje enviado por un canal P2P puede ser consumido por a lo sumo por
una aplicación.
context P2P inv:
    self.reqEnvio.mensaje->forAll(m | m.reqConsumo->size() <= 1)

-- Una aplicación subscripta a un canal pub/sub tiene un registro de consumo
sobre dicho canal y la fecha de subcripción es mayor a la fecha del registro
de consumo.
context Aplicacion inv:
    self.subscripcion->forAll(s | s.pubSub.regConsumo.aplicacion-
>includes(self))

context ReqConsumo inv:
    self.aplicacion.subscripcion->forAll(s | self.canal = s.pubSub implies
self.fecha < s.fecha)

-- Los registros de envío de un mensaje corresponden a la misma aplicación.
context Mensaje inv:
    self.regEnvio->forAll(r1, r2 | r1.aplicacion = r2.aplicacion)
```

Problema 2 (25 puntos)

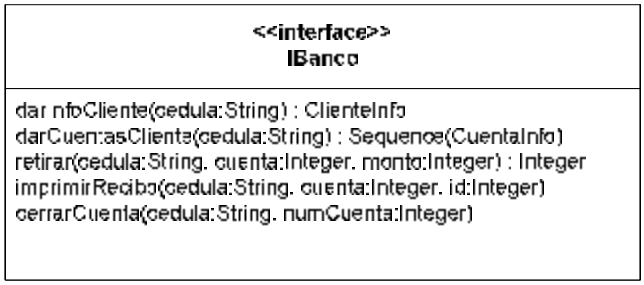
a) Ver slide 4 del juego de slides de diseño – visibilidad.

b.i)

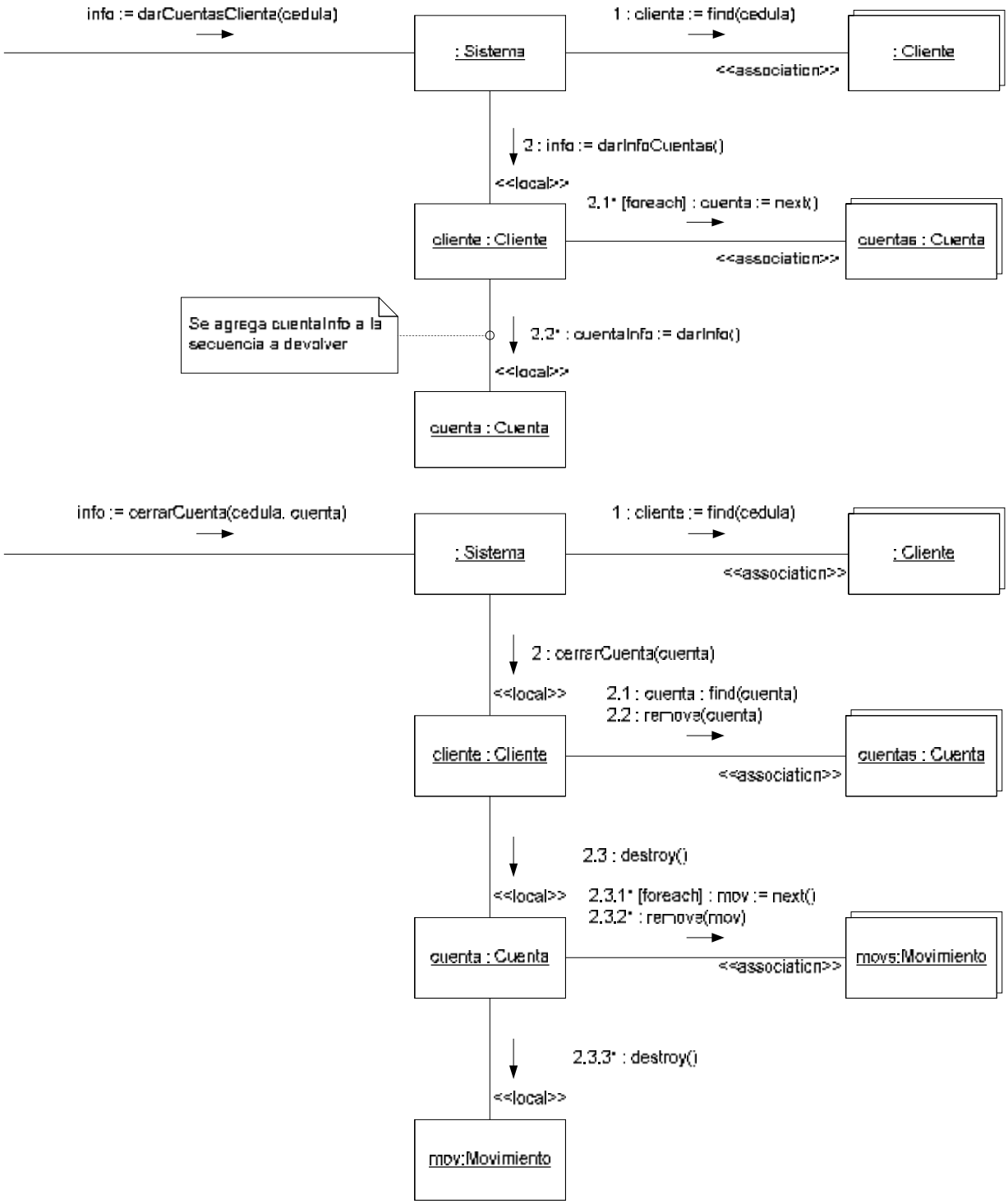



Nota: no es necesario pasar el id del movimiento porque es generado automáticamente, ni su fecha ya que se obtiene internamente en el sistema.

b.ii)

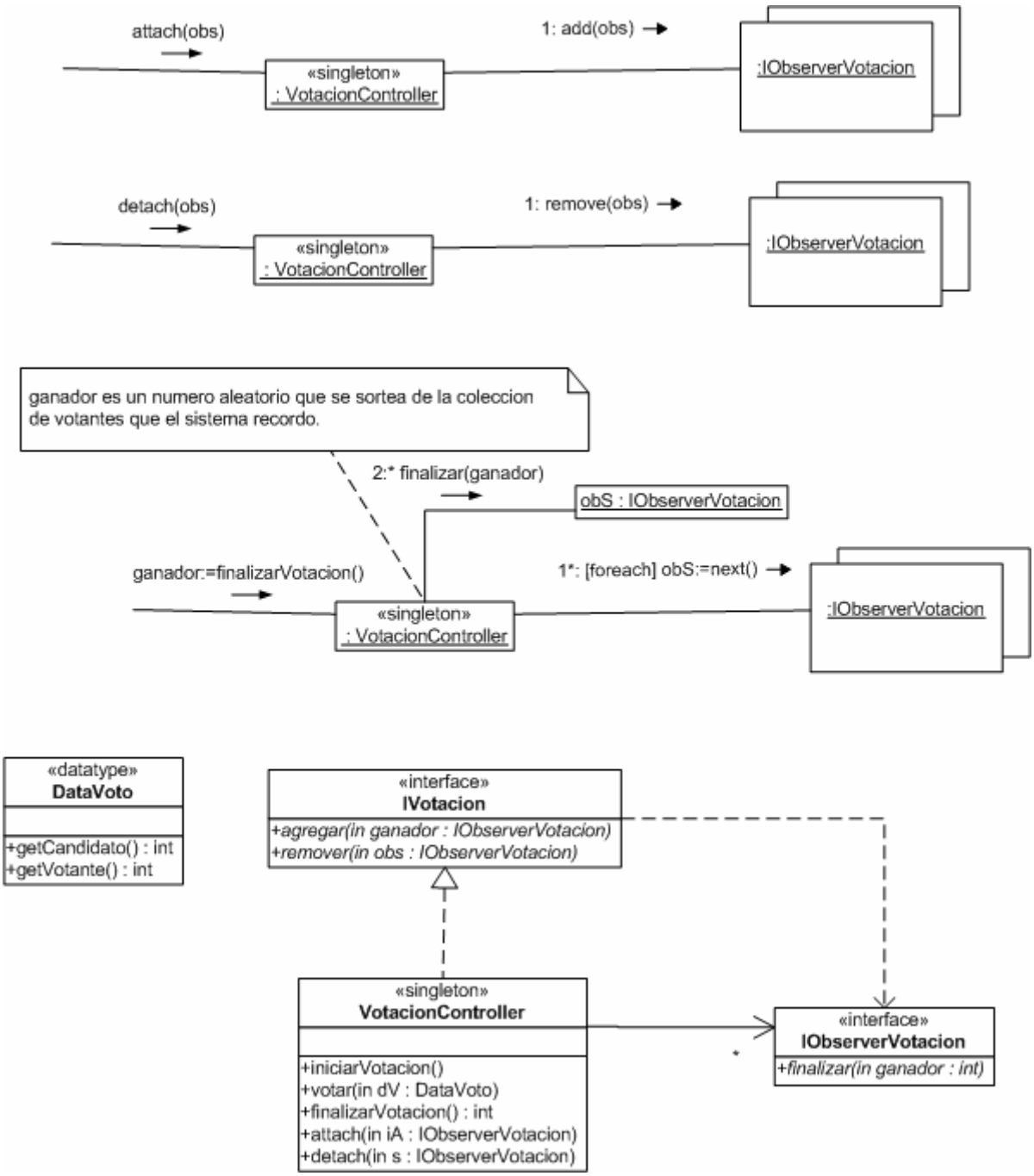

<<interface>>
IBanco
darInfoCliente(cedula:String) : ClienteInfo
darCuentasCliente(cedula:String) : Sequence(CuentaInfo)
retirar(cedula:String, cuenta:Integer, monto:Integer) : Integer
imprimirRecibo(cedula:String, cuenta:Integer, id:Integer)
cerrarCuenta(cedula:String, numCuenta:Integer)

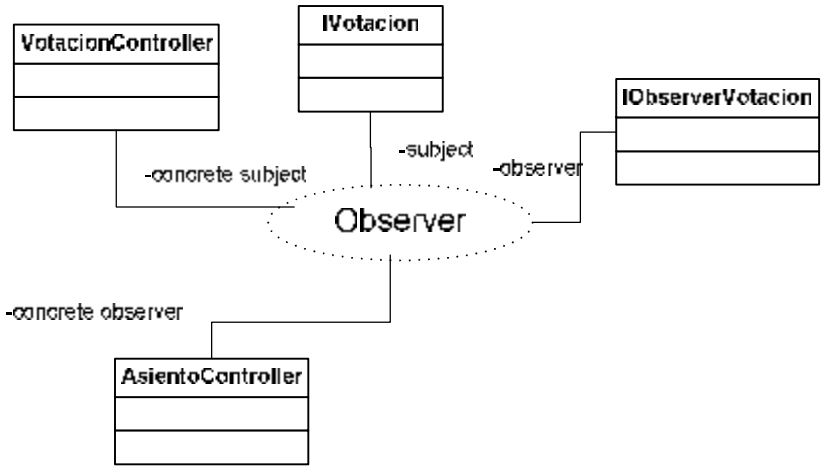
b.iii)


info := darCuentasCliente(cedula)
: Sistema
1 : cliente := find(cedula)
<<association>>
: Cliente
2 : info := darInfoCuentas()
<<local>>
2.1* [foreach] : cuenta := next()
cliente : Cliente
cuentas : Cuenta
<<association>>
Se agrega cuentaInfo a la secuencia a devolver
2.2* : cuentaInfo := darInfo()
<<local>>
cuenta : Cuenta
info := cerrarCuenta(cedula, cuenta)
: Sistema
1 : cliente := find(cedula)
<<association>>
: Cliente
2 : cerrarCuenta(cuenta)
<<local>>
2.1 : cuenta : find(cuenta)
2.2 : remove(cuenta)
cliente : Cliente
cuentas : Cuenta
<<association>>
2.3 : destroy()
<<local>>
2.3.1* [foreach] : mov := next()
2.3.2* : remove(mov)
cuenta : Cuenta
movs:Movimiento
<<association>>
2.3.3* : destroy()
<<local>>
mov:Movimiento

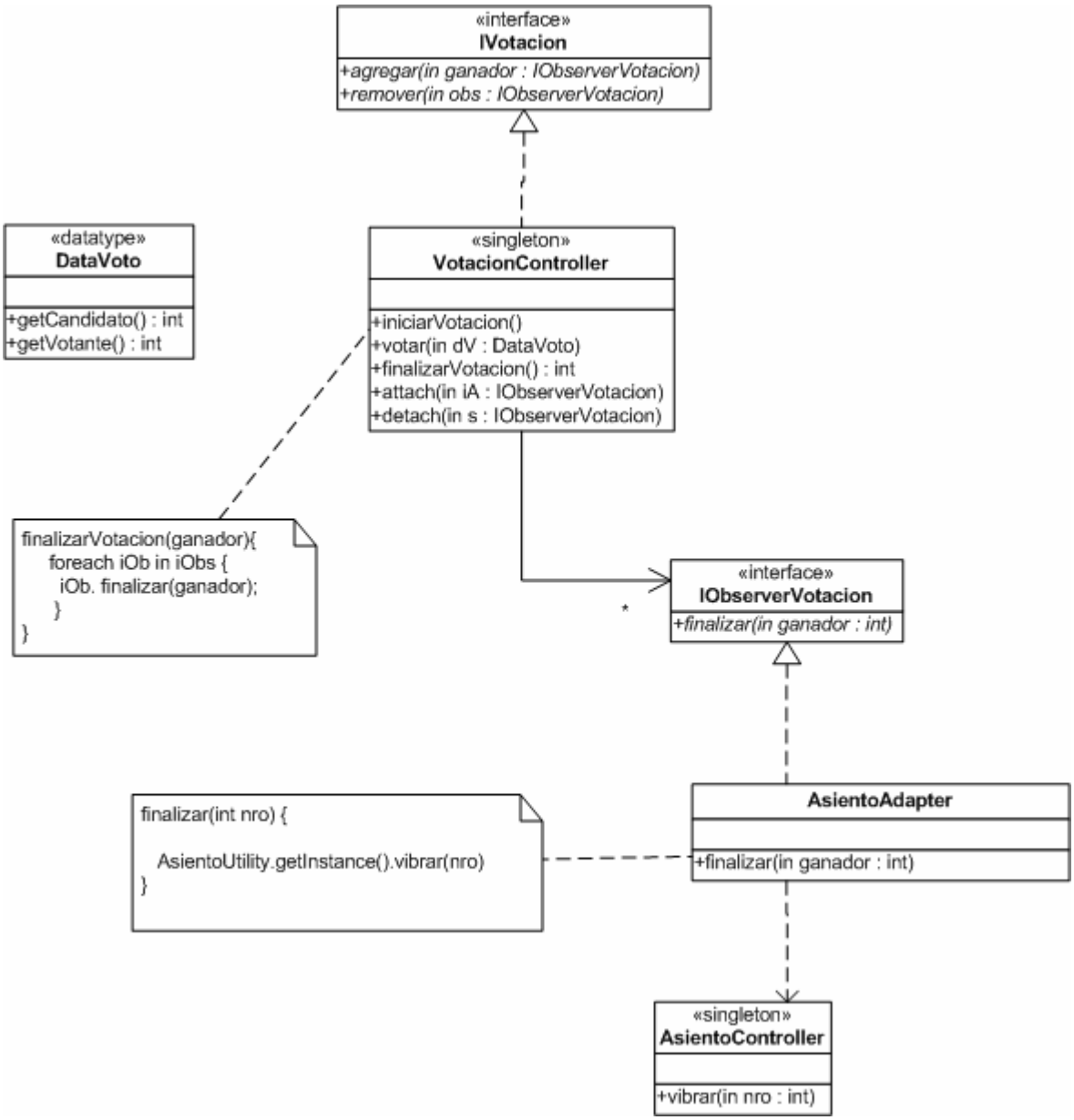
Problema 3 (25 puntos)

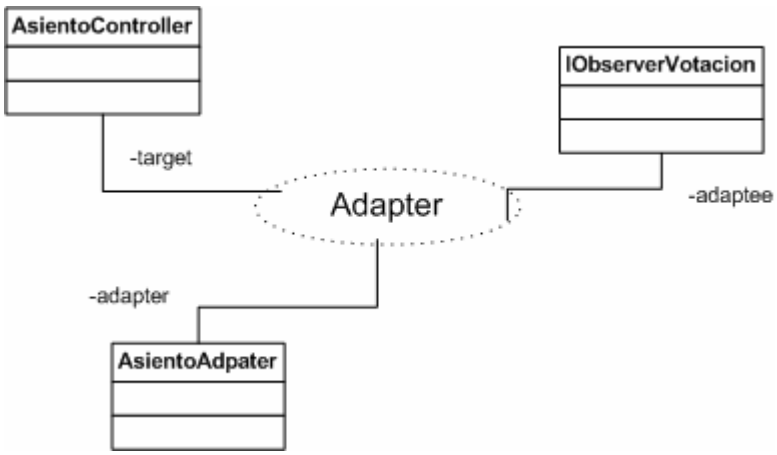
a) Se necesita que otros componentes sean notificados de manera desacoplada, por lo cual un patrón ideal para esto es el observer. Por lo cual la clase Votacion controller necesita dos operaciones attach y detach, la operación finalzarVotacion actua como el notify.






b) Usando el observer de la parte anterior es que se continúa con el diseño. La idea es que cuando se invoque finalizar, el observer deberá invocar la operación vibrar de la clase AsientoController, como dicha clase no se puede modificar se necesita adaptar la interfase, utilizando el patrón adapter.


«interface»
IVotacion
+agregar(in ganador : IObserverVotacion)
+remover(in obs : IObserverVotacion)
«datatype»
DataVoto
+getCandidato() : int
+getVotante() : int
«singleton»
VotacionController
+iniciarVotacion()
+votar(in dV : DataVoto)
+finalizarVotacion() : int
+attach(in iA : IObserverVotacion)
+detach(in s : IObserverVotacion)
finalizarVotacion(ganador){
foreach iOb in iObs {
iOb. finalizar(ganador);
}
}
«interface»
IObserverVotacion
+finalizar(in ganador : int)
*
finalizar(int nro) {
AsientoUtility.getInstance().vibrar(nro)
}
AsientoAdapter
+finalizar(in ganador : int)
«singleton»
AsientoController
+vibrar(in nro : int)


AsientoController
IObserverVotacion
-target
Adapter
-adaptee
-adapter
AsientoAdpater

Problema 4 (25 puntos)

a) Ver teórico.

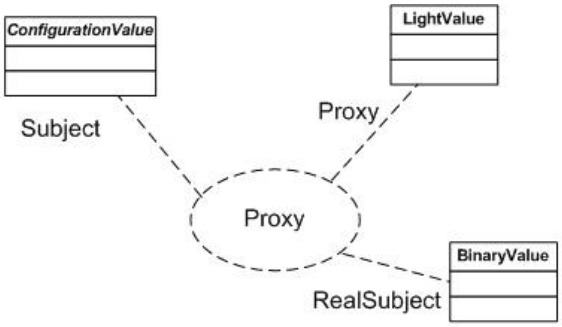
b.i)

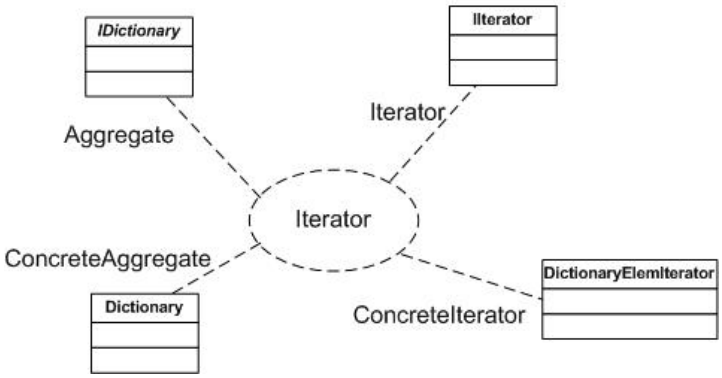
```
//IDictionary.hh
/*
 * La representación más basica de la estructura que maneja los
 * valores de configuración es que esta
 * sea un diccionario que guarda el valor de configuración con la
 * clave del nombre de la pantalla de configuracion
 * concatenado con el nombre del valor de configuración.
 * Ejemplo: estructura->add("pantalla1" + "rutaEnDisco","/home/.../");
 */
class IDictionary {
      public:
            virtual void add(IKey * key, ICollectible * value) = 0;
            virtual ICollectible * remove(IKey * key) = 0;
            virtual ICollectible * find(IKey * key) = 0;
            virtual bool member(IKey * key) = 0;
            virtual bool isEmpty() = 0;
            virtual IIterator * getElemIterator();
            virtual IIterator * getKeyIterator();
            virtual ~IDictionary();
}

//IKey.hh
class IKey {
      public:
            virtual bool equals(IKey * key);
            virtual ~IKey();
}

//ICollectible.hh
class ICollectible {
      public:
            virtual ~ICollectible();
}

//IIterator.hh
class IIterator {
      public:
            virtual void next() = 0;
            virtual ICollectible * current() = 0;
            virtual bool hasCurrent() = 0;
            virtual ~IIterator();
}

//ICollection.hh
class ICollection{
      public:
            virtual void add(ICollectible * elem) = 0;
            virtual void remove(ICollectible * elem) = 0;
            virtual bool member(ICollectible * elem) = 0;
            virtual bool isEmpty() = 0;
            virtual ~ICollection();
}
```

b.ii)

```
//InstallationManager.hh
class InstallationManager {
      private:
            IDictionary * estructura;

      public:
            InstallationManager();
            void makeInstall();
            ~InstallationManager(); //No se implementa.
}

//InstallationManager.cc
InstallationManager::InstallationManager() {
      estructura = new Dictionary(); //implementación de la interfaz
                                     //IDictionary.
}

void InstallationManager::makeInstall(){
      IIterator * it = estructura->getIterator();
      ICollection * col = new LinkedList(); //Implementación de la
                                            //interfaz ICollection.
      ConfigurationValue * conf; //se asume que implementa
                                 //ICollectible.
      while (it->hasCurrent()){
            conf = (ConfigurationValue *) it->current();
            conf->prepare();
            col->add(conf);
            it->next();
      }
      delete it;
      MainInstaller::install(col);
}
```

b.iii)


ConfigurationValue
LightValue
Subject
Proxy
Proxy
BinaryValue
RealSubject


IDictionary
IIterator
Aggregate
Iterator
Iterator
ConcreteAggregate
Dictionary
DictionaryElemIterator
ConcreteIterator