Programación 4

EXAMEN DICIEMBRE 2008

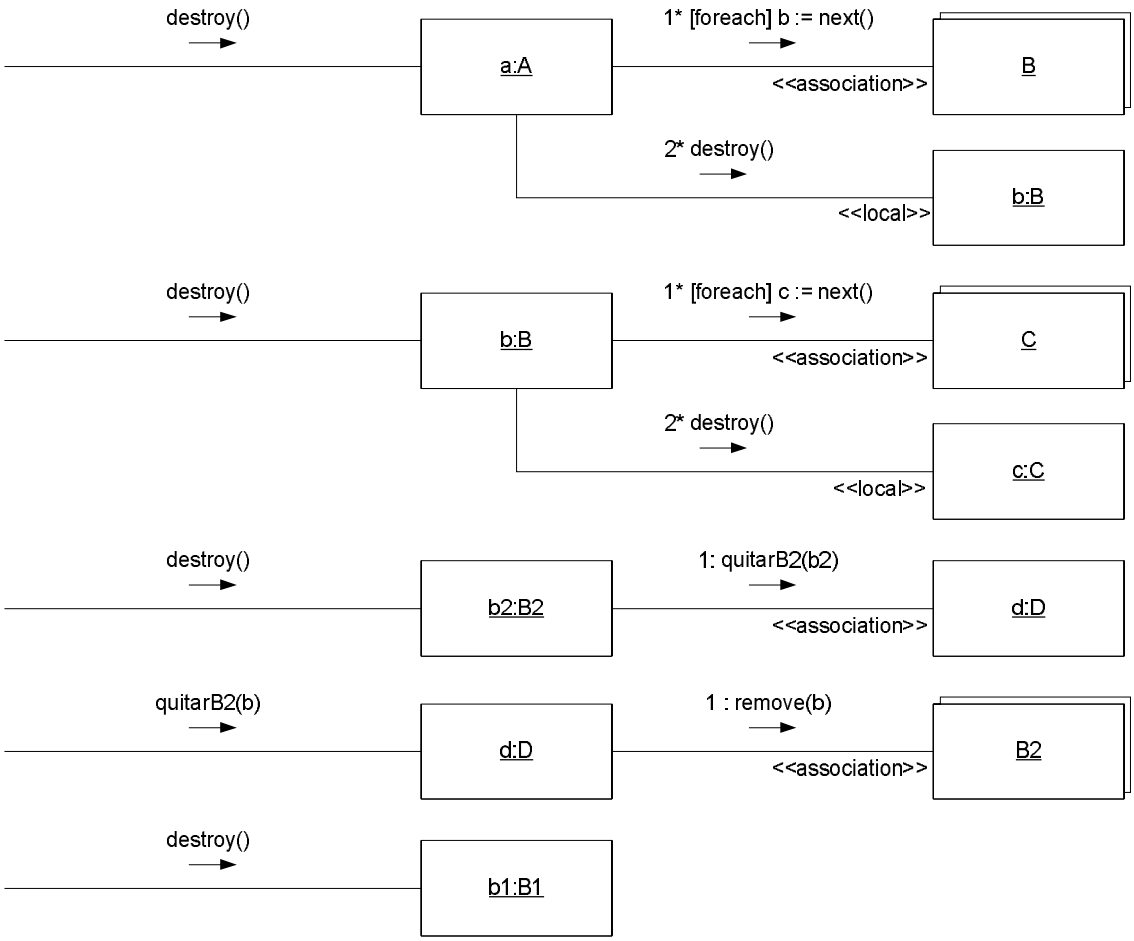
SOLUCIÓN

Problema 1 (35 puntos)

a) Enumere y describa muy brevemente los criterios para la asignación de responsabilidades vistos en el curso, cada uno en no más de una oración.

- Expert: Responsabilizar a quién tenga la información necesaria.
- Creator: A quién responsabilizar de la creación de un objeto.
- Bajo Acoplamiento: Evitar que un objeto interactúe con demasiados objetos.
- Alta Cohesión: Evitar que un objeto haga demasiado trabajo.
- No Hables con Extraños: Asegurarse que un objeto realmente delega trabajo.
- Controller: A quién responsabilizar de ser el controlador.

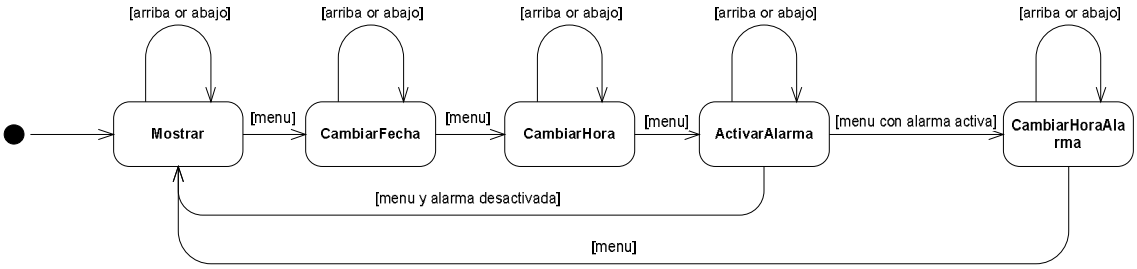
b)



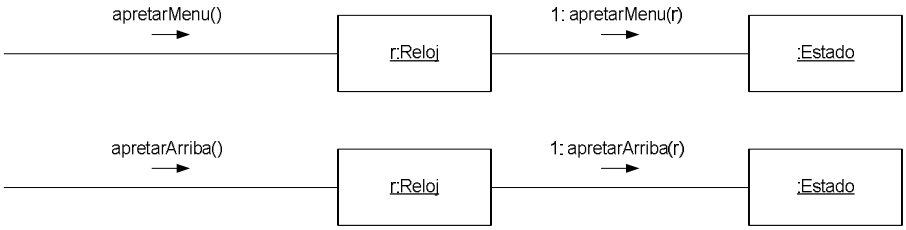
c)

i. Realice el/los diagramas de comunicación para las operaciones apretarMenu() y apretarArriba() de la clase Reloj.

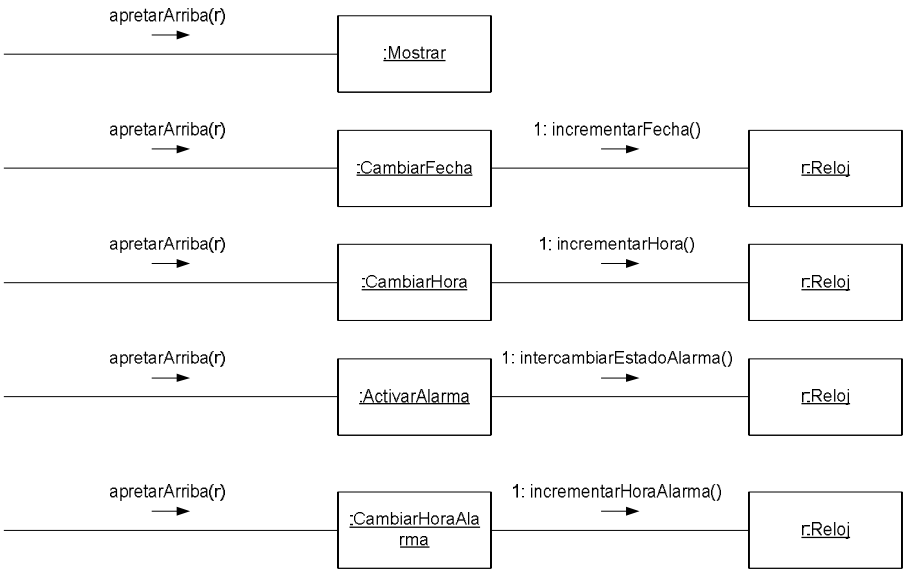
A modo de aclaración (es decir que no es necesario incluirlo), la máquina de estados que se pide diseñar es la siguiente:



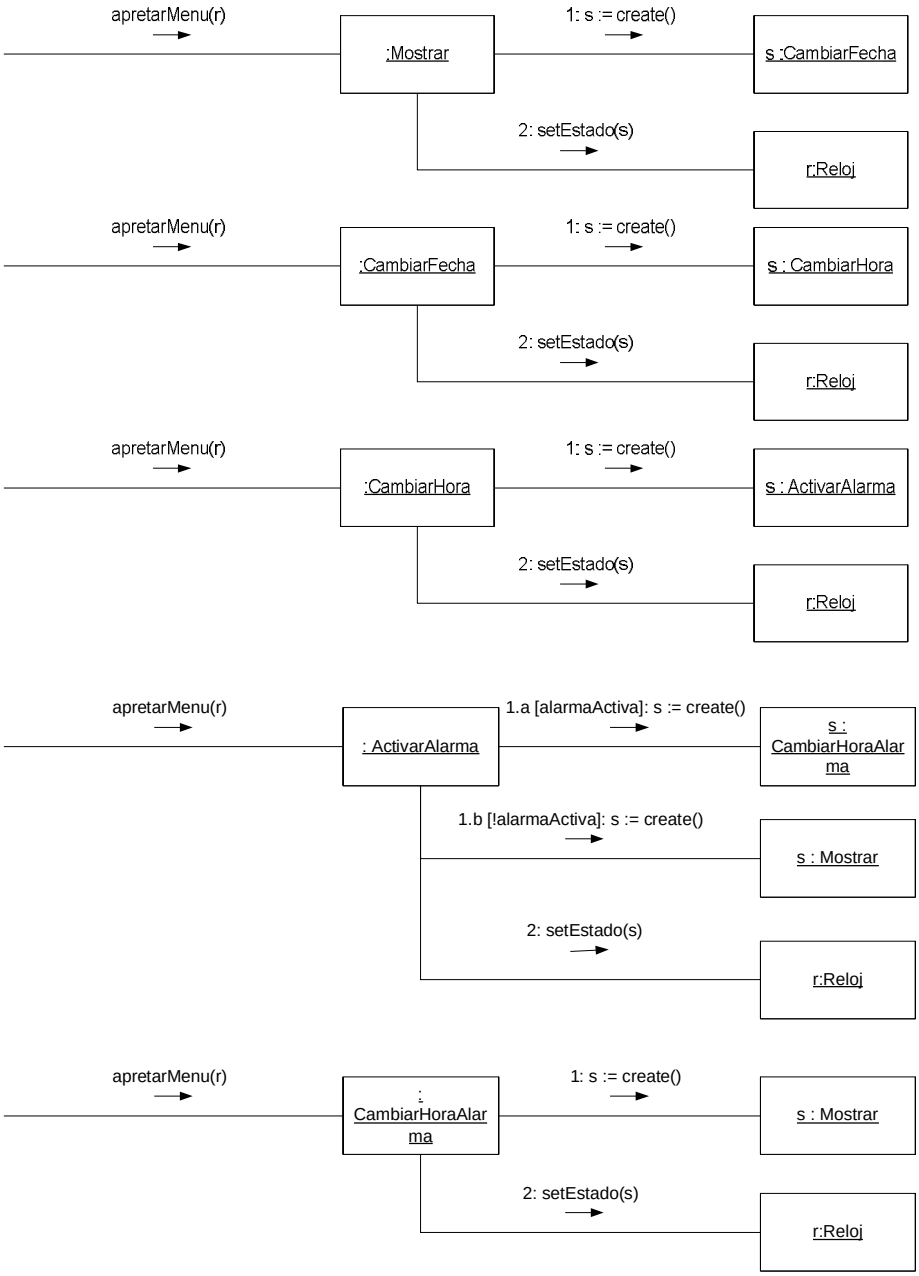
La mejor solución es utilizando el patrón State y delegando la responsabilidad de realizar las operaciones a los estados:



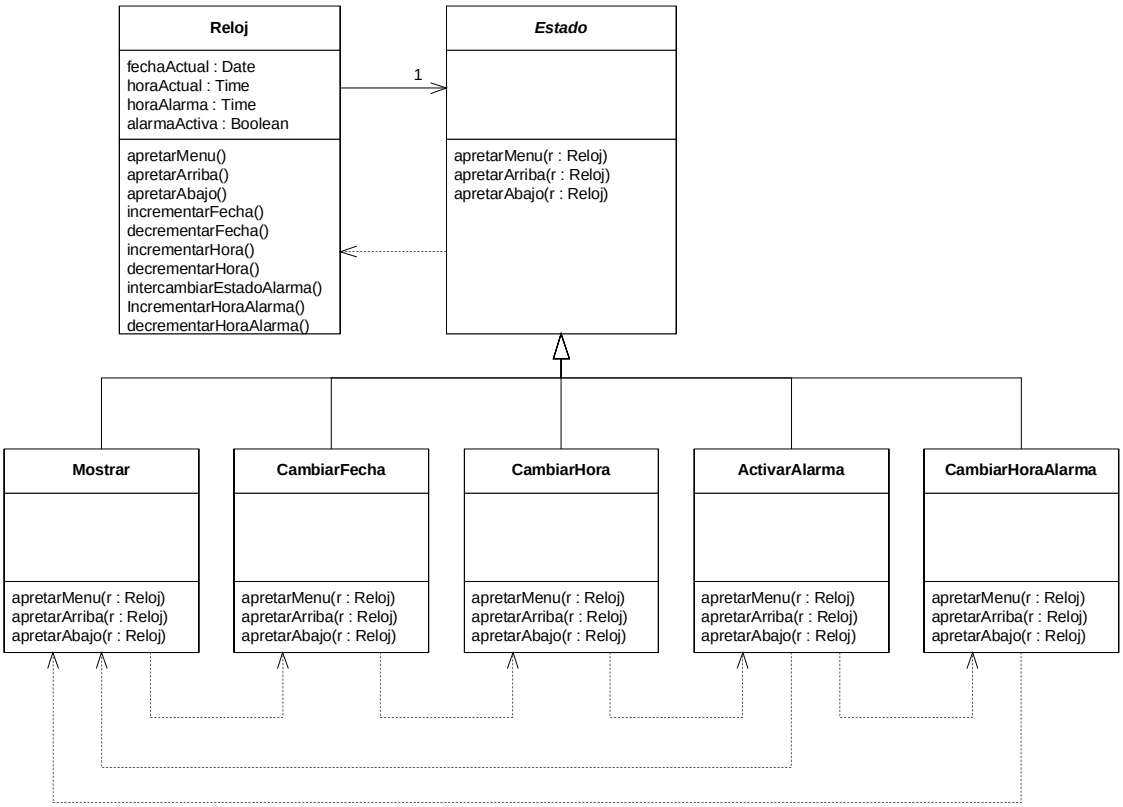
A continuación se muestran los diagramas para la operación apretarArriba de los distintos estados:



A continuación se muestran los cambios de estado, esta solución se basa en la vista en el teórico:



ii. Realice un diagrama de clases de diseño completo de su solución.



iii. Si utilizó algún patrón especifique su nombre, clases participantes y roles.

Patron: State

Clases particpantes y roles:

- Contexto: Reloj
- Estado: Estado
- Estados concretos: Mostrar, CambiarFecha, CambiarHora, ActivarAlarma, CambiarHoraAlarma.

Problema 2 (35 puntos)

a)

i) Ver teórico de conceptos básicos, diapositiva 62.

En C++:

```
Jornalero *j1 = null;          // void
Jornalero *j2 = new Jornalero(); // attached
```

ii) Ver teórico de conceptos básicos, diapositiva 64.

En C++:

```
Jornalero *j = new Jornalero();
```

TipoEstatico(j) = TipoDinamico(j) = Jornalero

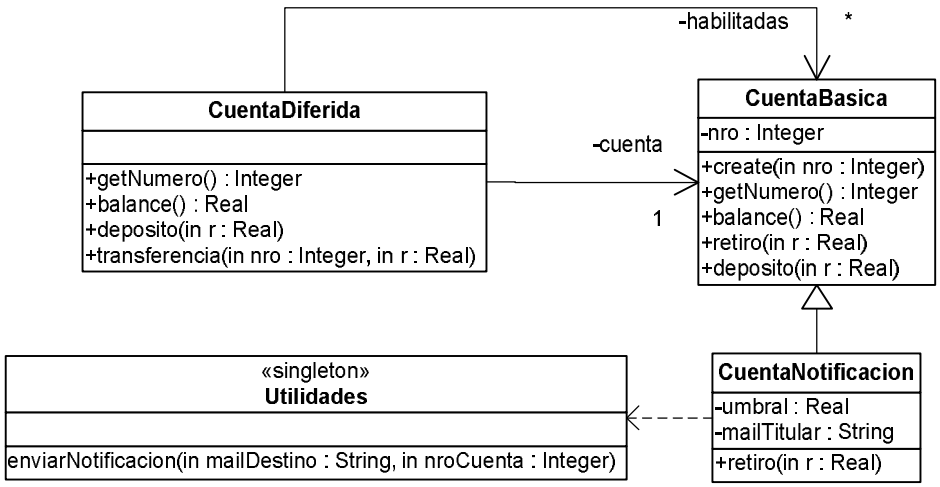
Empleado *e = new Jornalero();

TipoEstatico(e) = Empleado

TipoDinamico(e) = Jornalero

b)

i) Realizar un DCD completo del problema planteado sin modificar las clases `CuentaBasica` y `Utilidades`.



Una cuenta diferida "no es" una cuenta básica dada la diferencia entre depósito y transferencia, por otro lado la cuenta con notificación sí puede verse como una especialización de cuenta básica.

i. Implementar en C++ el diseño realizado en i) teniendo en cuenta lo siguiente:

- Incluir todos los .h
- Incluir únicamente los .cc de los dos nuevos tipos de cuenta
- Incluir constructores pero no destructores
- Asumir que existen implementaciones de la interfaces `ICollection` e `IIterator` usuales y también se cuenta con una implementación de `String`
- No incluir directivas al preprocesador en el código.

```
// CuentaBasica.h
class CuentaBasica : public ICollectible
{
private:
    int nro;
public:
    CuentaBasica(int);
    virtual int getNumero();
    virtual float balance();
    virtual void retiro(float);
    virtual void deposito(float);
```

```
};

// CuentaNotifiacion.h
class CuentaNotificacion : public CuentaBasica
{
private:
    String mailTitular;
    float umbral;
public:
    CuentaNotificacion(int, String, float);
    void retiro(float);
};

// CuentaNotifiacion.cc
CuentaNotificacion::CuentaNotificacion(int nro, String mail, float umbral) :
     CuentaBasica(nro), mailTitular(mail), umbral(umbral)
{
}

void CuentaNotificacion::retiro(float f)
{
    this->CuentaBasica::retiro(f);
    if (this->balance() < this->umbral)
    {
            Utilidades::getInstance()->enviarNotificacion(
                    this->mailTitular, this->getNumero());
    }
}

// CuentaDiferida.h
class CuentaDiferida
{
private:
    ICollection *cuentasHabilitadas;
    CuentaBasica *cuenta;
public:
    CuentaDiferida(int);
    int getNumero();
    float balance();
    void deposito(float);
    void transferencia(int, float);
    void agregarCuenta(CuentaBasica*);
    void quitarCuenta(CuentaBasica*);
};

// CuentaDiferida.cc
```

```
CuentaDiferida::CuentaDiferida(int nro) :
    cuenta(new CuentaBasica(nro)), cuentasHabilitadas(new List())
{
}

int CuentaDiferida::getNumero()
{
    return this->cuenta->getNumero();
}

float CuentaDiferida::balance()
{
    return this->cuenta->balance();
}

void CuentaDiferida::deposito(float f)
{
    this->cuenta->deposito(f);
}

void CuentaDiferida::transferencia(int nro, float f)
{
    IIterator *i = this->cuentasHabilitadas->getIterator();

    while (i->hasCurrent())
    {
            CuentaBasica *c = (CuentaBasica*)i->current();
            if (c->getNumero() == nro)
            {
                    this->cuenta->retiro(f);
                    c->deposito(f);
                    break;
            }

            i->next();
    }

    delete i;
}

void CuentaDiferida::agregarCuenta(CuentaBasica* c)
{
    this->cuentasHabilitadas->add(c);
}

void CuentaDiferida::quitarCuenta(CuentaBasica* c)
```

```
{
    this->cuentasHabilitadas->remove(c);
}

// Utilidades.h
class Utilidades
{
private:
    static Utilidades *instance;
    Utilidades();
public:
    static Utilidades* getInstance();
    void enviarNotificacion(String, int);
};
```

Problema 3 (30 puntos)

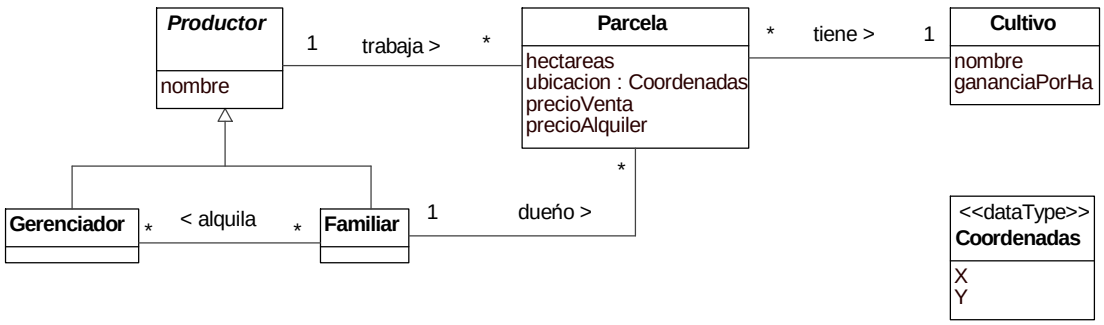
a) i.

Unicidad de Atributos: un atributo tiene un valor único dentro del universo de instancias de un mismo tipo.
Dominio de Atributos: el valor de un atributo pertenece a cierto dominio.
Integridad Circular: no puede existir circularidad en la navegación.
Atributos Calculados: el valor de un atributo es calculado a partir de la información contenida en el dominio.
Reglas de Negocio: invariante que restringe el dominio del problema.

ii.

Set: conjunto sin orden y sin elementos repetidos.
Sequence: set ordenado.
Bag: conjunto sin orden con elementos repetidos.

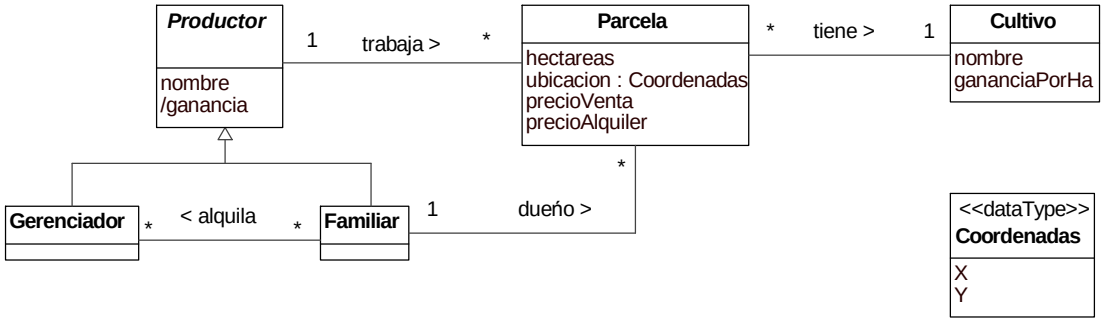
b) i.



Restricciones:

- Un productor familiar alquila al gerenciador parcelas que sean de su propiedad.
- Un productor familiar trabaja aquellas parcelas de su propiedad que no ha dado en alquiler.
- Las coordenadas de las parcelas son únicas.

ii.



Restricciones:

-- La ganancia de un productor es la suma de las ganancias para cada parcela

```
context Productor inv:
self.ganancia = self.parcela→collect(hectareas*cultivo.gananciaPorHa)→sum()
```