Programación 4

Implementación

Patrones de Diseño

Contenido

- Singleton
- State
- Observer
- Factory

Singleton

Singleton - Ejemplo

Singleton
-instancia : Singleton
-Singleton() +getInstancia() : Singleton +operacion()

Singleton – Código

```
// Singleton.h
class Singleton {
private:
	static Singleton * instancia;
	Singleton();
public:
	static Singleton * getInstancia();
	void operacion();
};
```

Singleton – Código (2)

```
// Singleton.cpp
#include "Singleton.h"
Singleton * Singleton::instancia = NULL;
Singleton::Singleton() {...}
Singleton * Singleton::getInstancia() {
   if (instancia == NULL)
      instancia = new Singleton();
   return instancia;
}
void Singleton::operacion() {...}
```

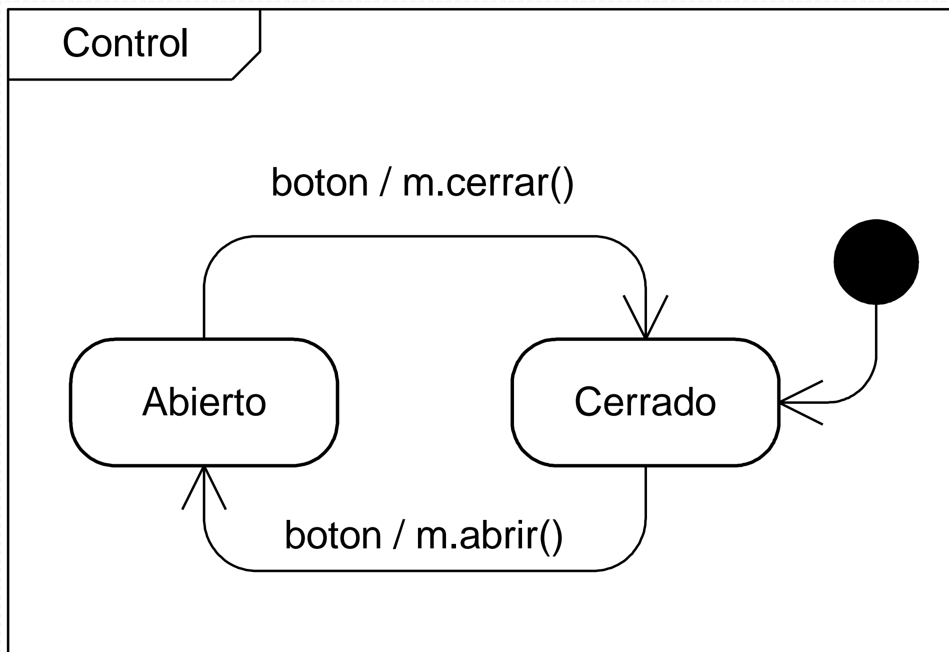
Singleton – Código (3)

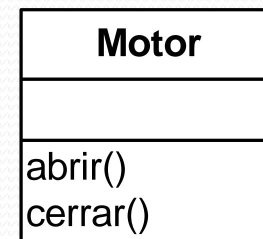
```
// Ejemplo de uso
#include "Singleton.h"
int main() {
      Singleton * ms;
      ms = Singleton::getInstancia();
      ms -> operacion();
      return 0;
}
```

State

State – Ejemplo

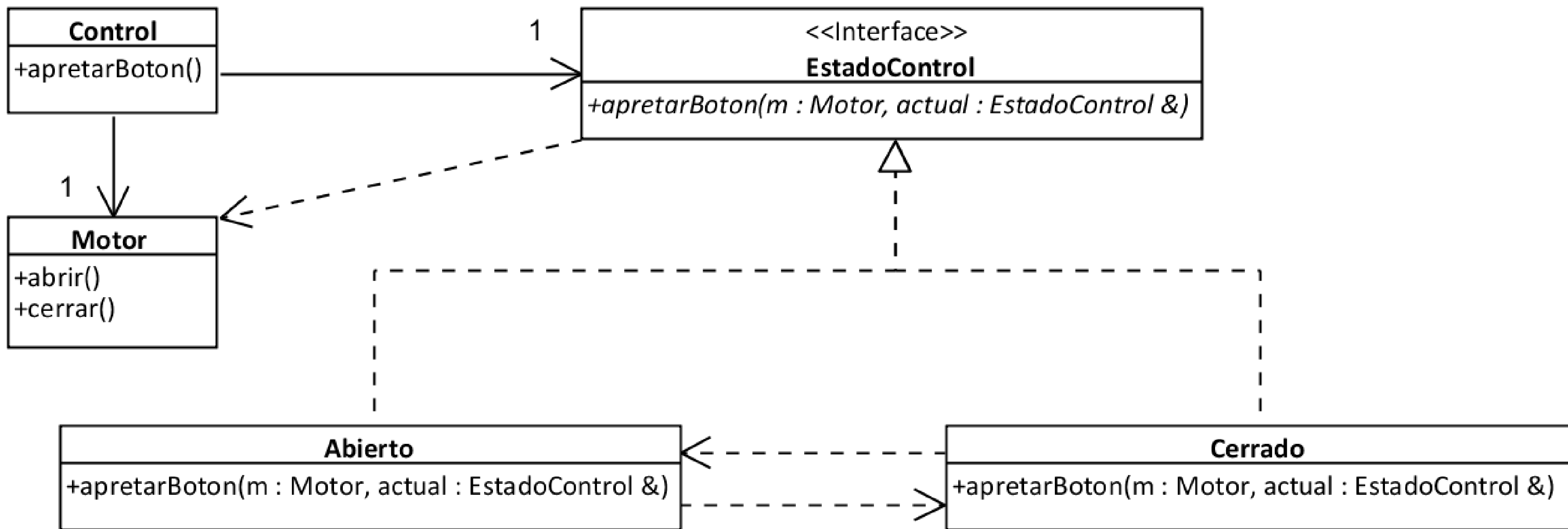
Puerta automática controlada por un control remoto de un solo botón:





`Motor` maneja el motor que abre y cierra la puerta

State – Ejemplo (2)



State – Código

```
// EstadoControl.h
#include "Motor.h"

class EstadoControl {
public:
  virtual void boton(Motor * m, EstadoControl *&e) = 0;
};
```

State – Código (2)

```
// Abierto.h

#include "EstadoControl.h"

class Abierto : public EstadoControl {
public:
  void boton(Motor * m, EstadoControl *&e);
};
```

State – Código (3)

```cpp
// Abierto.cpp

#include "EstadoControl.h"
#include "Motor.h"
#include "Abierto.h"
#include "Cerrado.h"

void Abierto::boton(Motor * m, EstadoControl *&e) {
      // Cerramos la puerta
      m -> cerrar();

      // Pasamos al proximo estado
      e = new Cerrado();
}
```

State – Código (4)

```
// Control.h

#include "EstadoControl.h“
#include "Motor.h"

class Control {
private:
	EstadoControl * estadoActual;
	Motor * motor;
public:
	Control();
	void apretarBoton();
};
```

State – Código (5)

```
#include "Control.h"
#include "Cerrado.h"
#include "EstadoControl.h"

Control::Control() {
      this->estadoActual = new Cerrado();
}

void Control::apretarBoton() {
  EstadoBoton *viejo = estadoActual;
  // cambia estadoActual
  estadoActual->boton(motor, estadoActual);

  if(viejo != estadoActual)
    delete viejo; // borrar viejo estado
}
```

State – Código (6)

```
// Ejemplo de uso
#include "Control.h"
int main() {
	Control * c = new Control();
	// Apretamos el boton (deberia ABRIRSE)
	c->apretarBoton();
	// Apretamoe el boton (deberia CERRARSE)
	c->apretarBoton();
	return 0;
}
```

State – Alternativas

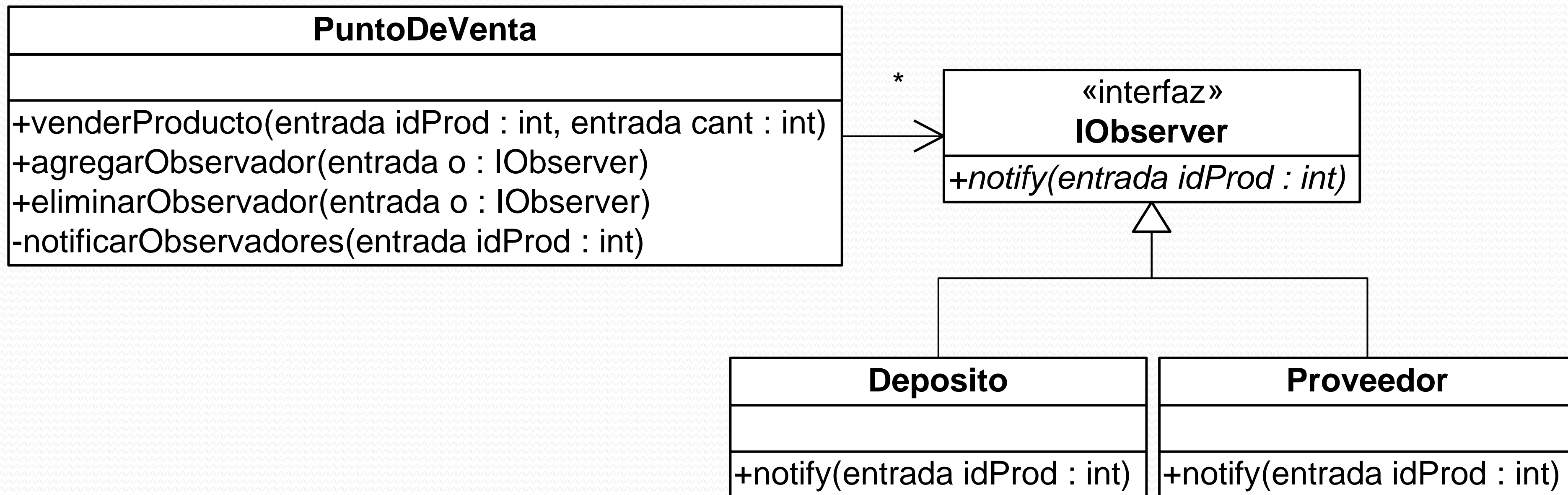
- Cuando el contexto pasa la información necesaria a los estados, éstos devuelven el nuevo estado y no dependen del contexto.
- Cuando el contexto se pasa a sí mismo a los estados, éstos pueden modificar directamente el estado actual del contexto y quedan dependientes del contexto.

Observer

Observer – Ejemplo

Un puesto de venta tiene varios productos para vender, de los cuales se necesita controlar su stock. Cuando el stock de alguno de ellos llegue a cero, deberá avisarse al depósito (para que se preparen a recibir los nuevos productos) y a los proveedores (para que provean de más stock).

Observer – Ejemplo (2)



Observer – Código

```
// PuntoDeVenta.h

#include “IObserver.h“
class PuntoDeVenta {
private:
   set<Producto *> productos;
   set<IObserver *> observers;
   void notificarObservadores(int idProd);
public:
   void venderProducto(int idProd, int cant);
   // Permite registrar un observador nuevo
   void agregarObservador(IObserver *o);
   // Permite desregistrar un observador
   void eliminarObservador(IObserver *o);
};
```

Observer – Código (2)

```
// PuntoDeVenta.cpp
void PuntoDeVenta::venderProducto(int idProd,int cant){
  if(productos[idProd]->getStock() < cant)
    notificarObservadores(idProd);
  // …
}

void PuntoDeVenta::notificarObservadores(int idProd){
  set<IObserver*>::iterator it;
  for(it = observers.begin(); it != observers.end(); ++it)
    it->notify(idProd);
}
```

Observer – Código (3)

```
// PuntoDeVenta.cpp (cont.)

void PuestoDeVenta::agregarObservador(IObserver *o){
   observers->insert(o);
}

void PuestoDeVenta::eliminarObservador(IObserver *o){
   observers->erase(o);
}
```

Observer – Código (4)

```
// Deposito.h

class Deposito : public IObserver {
private:
   ...
public:
   void notify(int idProd);
};
```

```
// Deposito.cpp

void Deposito::notify(int idProd) {
   // hacer lugar para recibir
   // nuevo stock de 'idProd'
}
```

Observer – Código (5)

```
// Proveedor.h

class Proveedor : public IObserver {
private:
   ...
public:
   void notify(int idProd);
};
```

```
// Proveedor.cpp

void Proveedor::notify(int idProd) {
   // pedir más unidades del
   // producto 'idProd'
}
```

Observer – Código (6)

```
// Ejemplo de uso
int Main() {
   PuntoDeVenta * pdv = ...
   Deposito * deposito = ...
   Proveedor * proveedor = ...

   // Registramos los observadores
   pdv -> agregarObservador(deposito);
   pdv -> agregarObservador(proveedor);

   // Vendemos un producto en cantidad
   // suficiente para agotar su stock
   pdv -> venderProducto(id, mucho);
}
```

Observer – Alternativas

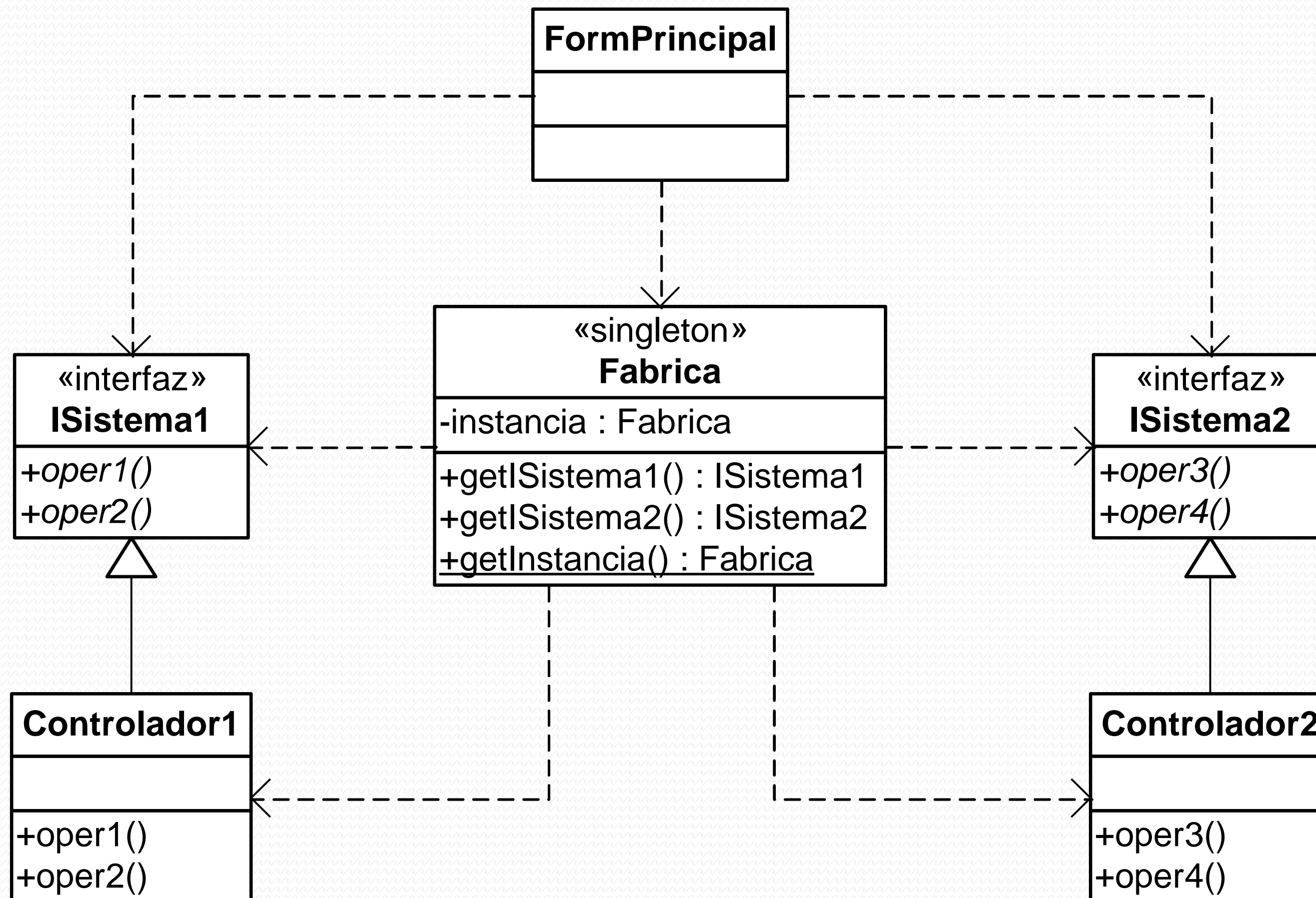
- Cuando el subject pasa la información necesaria a los observadores (como datos en los parámetros del *notify*), éstos no dependerán de él (como en el ejemplo).
- Cuando el subject se pasa a si mismo a los observadores (como *this* en el *notify*), éstos dependerán de él.

Factory

Factory – Ejemplo

Se tiene una fábrica de controladores que permite el acceso a éstos desde la presentación, sin que ésta quede acoplada a los controladores (mediante el uso de interfaces del sistema).

Factory – Ejemplo (2)



Factory – Alternativas

1. Que los controladores sean Singleton y simplemente sean accedidos mediante *getInstance()* desde la Fábrica.
2. Que los controladores no sean Singleton y que la Fábrica mantenga una referencia a cada uno de ellos controlando su unicidad.
3. Que los controladores no sean Singleton y que la Fábrica no mantenga referencias, sino que devuelva una nueva instancia del controlador cada vez.

Factory – Código

```
// ------------ Alternativa 1 ------------
ISistema1 Fabrica::getISistema1() {
   return Controlador1.getInstancia();
}

// ------------ Alternativa 2 ------------
ISistema1 Fabrica::getISistema1() {
   if (this->sistema1 == NULL)
      this->sistema1 = new Controlador1();
   return this->sistema1;
}

// ------------ Alternativa 3 ------------
ISistema1 Fabrica::getISistema1() {
   return new Controlador1();
}
```