Diseño Orientado a Objetos: Conceptos y Principios

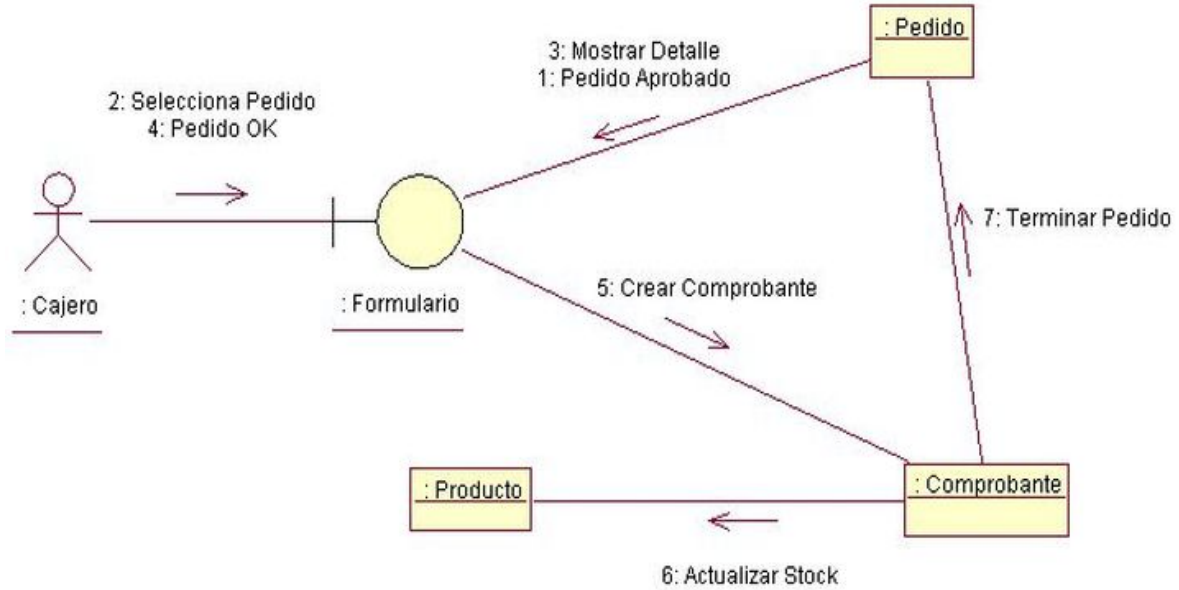
Julio Ariel Hurtado Alegría

23 de febrero de 2015

Contenido

Motivación

La orientación a objetos es una forma de implementar un programa combinando adecuadamente los principios de abstracción, modularidad, jerarquías, encapsulamiento, ocultamiento de la información para lograr una adecuada separación de preocupaciones.

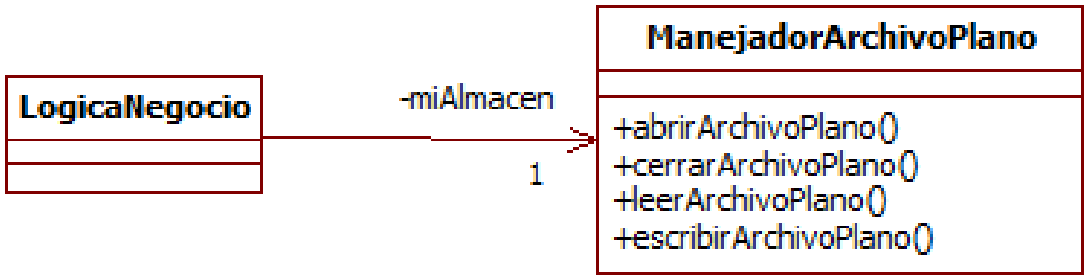


Motivación

- Conceptos: herencia, polimorfismo, encapsulamiento, ocultamiento de la información.
- Principios generales.
- Principios fundamentales.
- Patrones de Diseño.

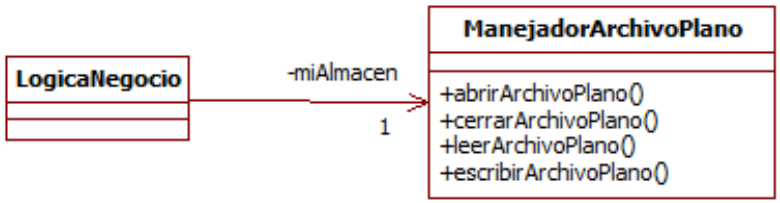
Programe para una interface y no para una implementación

Con este principio se busca desacoplar las partes de un sistema. Esto es, que un componente no dependa de la implementación de otros. Para ello el comportamiento usado puede ser encapsulado a través de una clase abstracta o una interface.



Programe para una interface y no para una implementación

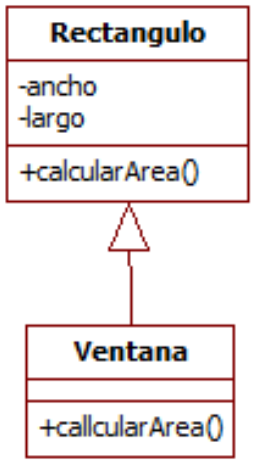
Con este principio se busca desacoplar las partes de un sistema. Esto es, que la lógica de ejecución no dependa de la implementación particular. Para ello el comportamiento usado puede ser encapsulado a través de clases abstractas o interfaces.



¿Qué pasará cuando se desea que la aplicación permita el almacenamiento usando en formato Texto o XML?

Favorezca la Composición a la Herencia

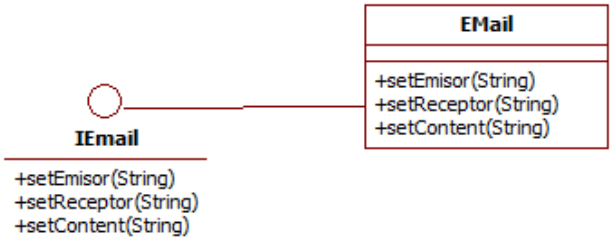
Nueva funcionalidad es obtenida ensamblando objetos de funcionalidad más simple, donde los objetos adquieren referencias a otros objetos.



¿Qué anda mal en este diseño? ¿Cómo podría mejorarse? ¿Cuál es la diferencia entre re-envío y delegación?

Principio Una Sola Responsabilidad

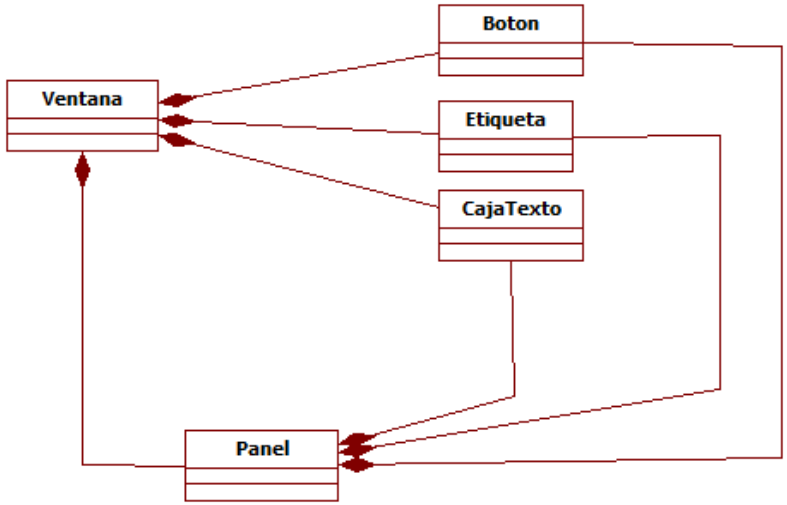
Todo objeto debe tener una sola responsabilidad y que esa resposabilidad debe ser completamente encapsulada en una clase. Una clase debe tener sólo una razón para cambiar.



¿Qué anda mal en este diseño? ¿Qué pasa si el contenido del e-mail es complejo?

Principio Abierto-Cerrado - OCP

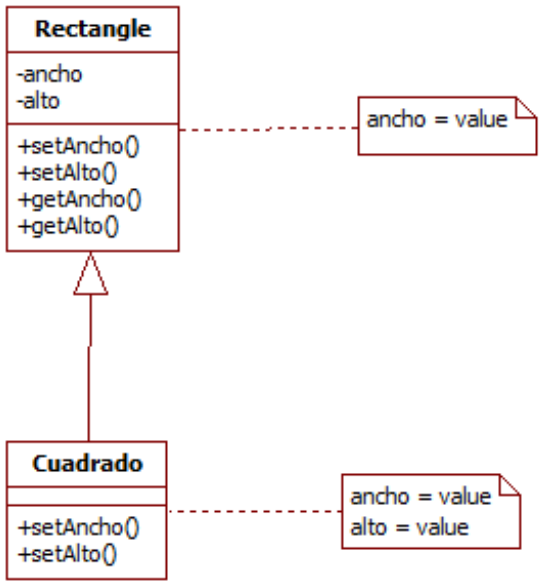
Las entidades de software, tales como clases deben ser abiertas a la extensión y cerradas a la modificación



¿Qué sucede si agregamos un panelWeb? ¿Que sucede si agregamos una Caja de Selección?

Principio de Substitución de Liskov

En un programa, si S es un subtipo de T entonces los objetos del tipo T pueden ser reemplazados con objetos de tipo S sin alterar el comportamiento del programa.



¿Qué pasa si se trabaja con un cuadrado pensando que es un rectángulo? ¿El cuadrado es en verdad de tipo rectángulo?

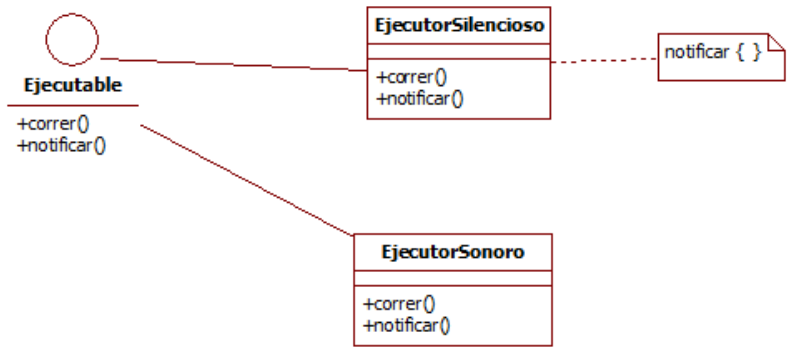
Principio de Substitución de Liskov - Diseño por Contrato

```
note
    description: "Simple bank accounts"
class
    ACCOUNT
feature -- Access
  balance: INTEGER
          -- Current balance
  deposit_count: INTEGER
          -- Number of deposits made since opening
      do
          ... As before ...
      end
feature -- Element change

  deposit (sum: INTEGER)
          -- Add `sum' to account.
      require
          non_negative: sum >= 0
      do
          ... As before ...
      ensure
          one_more_deposit: deposit_count = old deposit_count + 1

          updated: balance = old balance + sum
      end
```

```
feature {NONE} -- Implementation

    all_deposits: DEPOSIT_LIST
            -- List of deposits since account's opening.

invariant
    consistent_balance: (all_deposits /= Void) implies
            (balance = all_deposits . total)
    zero_if_no_deposits: (all_deposits = Void) implies
            (balance = 0)

end -- class ACCOUNT
```

Principio de Segregación de Interfaces

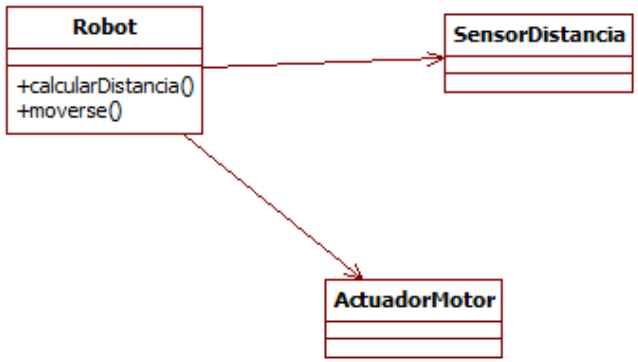
Módulos clientes no deberían ser forzados a implementar interfaces que ellos no usarán. Es preferible tener muchas interfaces pequeñas basadas en grupos de métodos asociadas a un módulo en particular, que tener una sóla interface grande.



¿Qué anda mal ahora? ¿Cómo puedo desagregar interfaces y tener más posibilidades de extensión?

Principio de Inversión de Dependencia

Modulos de alto nivel no dependen de módulos de bajo nivel. Ambos deberían depender de abstracciones, abstracciones no dependen de detalles. Detalles dependen de las abstracciones.



¿Qué anda mal ahora? ¿El robot como elemento abstracto depende de implementaciones estables?

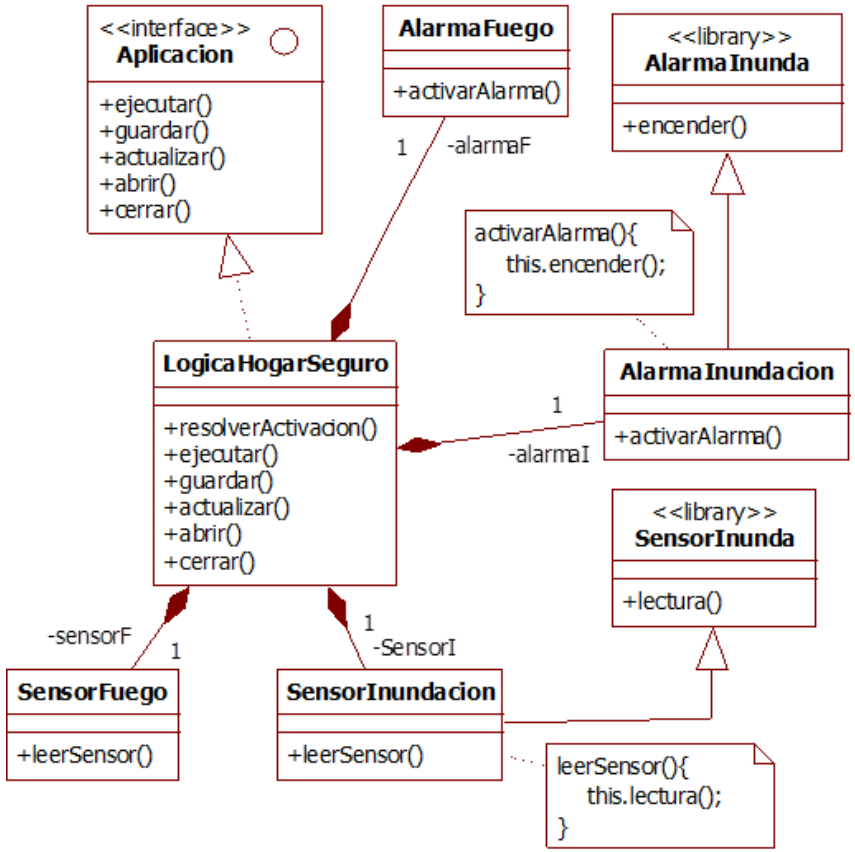
Causas Frecuentes de Rediseño

- Crear clases especificando una clase explícitamente. ¿Podemos tener una entidad creadora?
- Dependencia de métodos específicos. ¿Podemos generalizar y delegar?
- Dependencia de plataformas software y hardware. Podemos abstraer y la creación dejársela a otros.
- Dependencia de la representación o implementación. Debe programarse para una interface y no para una implementación.
- Dependencia de algortimos específicos. Deben abstraerse los métodos y encapsularse.
- Imposibilidad de alterar las clases convenientemente. Uso de la composición para empatar servicios.

Ejercicio

Considere el diseño analizado anteriormente, identifique la violación de principios de diseño y proponga un nuevo diseño que mejore la solución.
Muestre mediante un diagrama de secuencia como se puede agregar la alarma al programa sin alterar la programación de ninguna de las clases existentes.

Ejercicio


<<interface>>
Aplicacion
+ejecutar()
+guardar()
+actualizar()
+abrir()
+cerrar()
AlarmaFuego
+activarAlarma()
1
-alarmaF
<<library>>
AlarmaInunda
+encender()
activarAlarma(){
this.encender();
}
LogicaHogarSeguro
+resolverActivacion()
+ejecutar()
+guardar()
+actualizar()
+abrir()
+cerrar()
1
-alarmaI
AlarmaInundacion
+activarAlarma()
<<library>>
SensorInunda
+lectura()
-sensorF
1
1
-SensorI
SensorFuego
+leerSensor()
SensorInundacion
+leerSensor()
leerSensor(){
this.lectura();
}

Conclusión

Los principios de diseño orientado a objetos, le dan al software la posibilidad ser construido con un alto grado de flexibilidad lo que facilita la mantención de aplicaciones, el reuso de infraestructuras pasivas (librerías) y el reuso de infraestructuras activas (frameworks).

1. Aplicaciones -Control concreto sobre una aplicación concreta.
2. Librerías y Toolkits - Clases y métodos relacionados sin control de alguna aplicación.
3. Frameworks - Control concreto sobre una aplicación abstracta a ser definida en forma concreta, principio de Holliwood.

Diseño Orientado a Objetos: Conceptos y Principios

Julio Ariel Hurtado Alegría

23 de febrero de 2015