38

MÉTODO DE PLANTILLA

Este patrón se describió previamente en GoF95.

DESCRIPCIÓN

El patrón Template Method es uno de los patrones de diseño más simples y más utilizados en aplicaciones orientadas a objetos.

El patrón del método de plantilla se puede usar en situaciones en las que hay un algoritmo, algunos de cuyos pasos se pueden implementar de varias maneras diferentes. En tales escenarios, el patrón del método de plantilla sugiere mantener el esquema del algoritmo en un método separado denominado*método de plantilla*dentro de una clase, que puede denominarse*clase de plantilla*, omitiendo las implementaciones específicas de las porciones variantes (pasos que se pueden implementar de múltiples maneras diferentes) del algoritmo para diferentes subclases de esta clase.

ÉlModelola clase no necesariamente tiene que dejar la implementación a las subclases en su totalidad. En cambio, como parte de proporcionar el esquema del algoritmo, elModeloLa clase también puede proporcionar cierta cantidad de implementación que puede considerarse invariable en diferentes implementaciones. Incluso puede proporcionar una implementación predeterminada para las piezas variantes, si corresponde. Solo se implementarán detalles específicos dentro de diferentes subclases. Este tipo de implementación elimina la necesidad de código duplicado, lo que significa que se debe escribir una cantidad mínima de código.

Utilizando el lenguaje de programación Java, elModeloLa clase se puede diseñar de una de las dos formas siguientes.

Clase abstracta

Este diseño es más adecuado cuando elModeloclass proporciona solo el esquema del algoritmo sin ninguna implementación predeterminada para sus partes variantes. Suponiendo que los diferentes pasos del algoritmo se pueden convertir en métodos individuales:

- ÉlModeloEl método puede ser un método concreto, no abstracto, con llamadas a otros métodos que representan diferentes pasos del algoritmo.
- ÉlModeloLa clase puede implementar partes invariantes del algoritmo como un conjunto de métodos no abstractos.

- El conjunto de pasos variantes se puede diseñar como métodos abstractos. Se pueden proporcionar implementaciones específicas para estos métodos abstractos dentro de un conjunto de subclases concretas del método abstracto.Modeloclase.

En este diseño, elAbstractoLa clase declara métodos y cada una de las subclases implementa estos métodos de una manera específica sin alterar el esquema del algoritmo.

Clase de hormigón

Este diseño es más adecuado cuando elModeloclass proporciona, además del esquema del algoritmo, la implementación predeterminada para sus partes variantes. Suponiendo que los diferentes pasos del algoritmo se pueden convertir en métodos individuales:

- ÉlModeloEl método puede ser un método concreto, no abstracto, con llamadas a otros métodos que representan diferentes pasos del algoritmo.
- ÉlModeloLa clase puede implementar partes invariantes del algoritmo como un conjunto de métodos no abstractos.
- El conjunto de pasos variantes se puede diseñar como métodos no abstractos con la implementación predeterminada. Subclases de laModeloLa clase puede anular estos métodos para proporcionar implementaciones específicas sin alterar el esquema del algoritmo.

A partir de ambas estrategias de diseño, se puede ver que la implementación del patrón de plantilla se basa en gran medida en la herencia y la anulación de funciones. Por lo tanto, cada vez que se usa la herencia para implementar los detalles, se puede decir que el patrón del método de plantilla se usa en su forma más simple.

EJEMPLO

Diseñemos una aplicación para comprobar la validez de una determinada tarjeta de crédito. Para simplificar, consideremos solo tres tipos de tarjetas de crédito: Visa, MasterCard y Diners Club. La aplicación realiza una serie de validaciones sobre la información de la tarjeta de crédito ingresada. Cuadro 38.1enumera diferentes pasos en el proceso de validación de diferentes tarjetas de crédito.

Como se puede ver en la Tabla 38.1, algunos pasos del algoritmo de validación son los mismos en las tres tarjetas de crédito, mientras que otros son diferentes. El patrón del método de plantilla se puede aplicar en el diseño de este proceso.

Definamos un resumenTarjeta de créditoclase (Figura 38.1y Listado 38.1) con:

- ÉlModelométodoes válidaque describe el algoritmo de validación.
- Un conjunto de métodos concretos que implementan el Paso 1, el Paso 4 y el Paso 5 de la Tabla 38.1.
- Un conjunto de métodos abstractos designados para implementar el Paso 2, el Paso 3 y el Paso 6 de la Tabla 38.1. Cabe señalar que incluso después de laSuma de verificaciónla validación es exitosa, no se puede garantizar que una determinada tarjeta de crédito sea válida. Es posible que la cuenta haya sido revocada o exceda el límite.

Cuadro 38.1 Diferentes pasos en el proceso de validación

Paso	*Controlar*	*Visa*	*tarjeta MasterCard*	*club de comensales*
1	Fecha de caducidad	> Hoy	> Hoy	> Hoy
2	Longitud	13, 16	dieciséis	14
3	Prefijo	4	51 a 55	30, 36, 38
4	Caracteres válidos	0 a 9	0 a 9	0 a 9
5	dígito de control algoritmo	Modo 10	Modo 10	Modo 10
6	cuenta en buen de pie	Usar personalizado API de Visa	Usar personalizado tarjeta MasterCard API	Usar personalizado club de comensales API

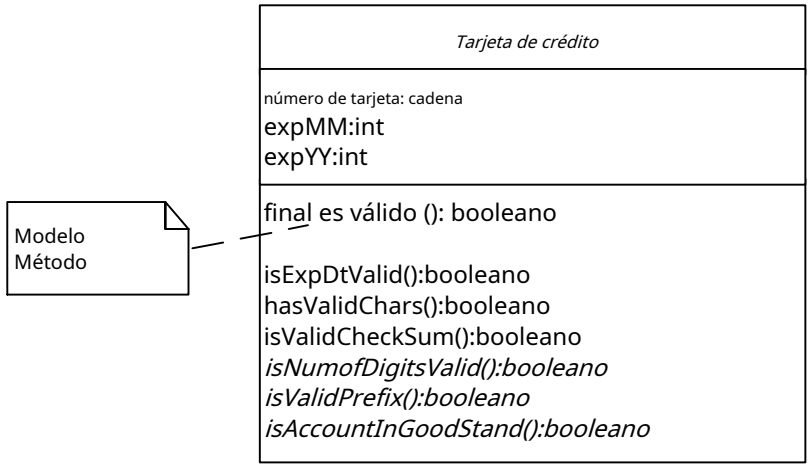

Figura 38.1Tarjeta de créditoClase de plantilla

Por lo tanto, se requiere una verificación con la compañía de la tarjeta de crédito (Visa, MasterCard, Diners Club) para asegurarse de que la cuenta esté al día. Este paso requiere una programación personalizada para interactuar con la base de datos de la compañía de tarjetas de crédito y se considera diferente para diferentes tipos de tarjetas de crédito. Por lo tanto, laesAccountInGoodStandEl método está diseñado como un método abstracto para ser implementado por diferentes subclases.

El método más significativo en el diseño es eles una plantilla válidamétodo. Este método invoca diferentes métodos diseñados para implementar diferentes pasos del algoritmo. Cabe señalar que elModelométodoes válidase especifica como un*fianal* método para evitar que las subclases lo anulen. Se espera que las subclases anulen solo los métodos abstractos para proporcionar una implementación específica y son*no* se supone que altera el esquema del algoritmo.

Listado 38.1 ResumenTarjeta de créditoClase

```
tarjeta de crédito de clase abstracta pública {
   protegido String cardNum; protegido
   int expMM, expYY;
   public CreditCard(String num, int expMonth, int expYear) {
      número de tarjeta = número;
      expMM = expMes;
      expYY = expaño;
   }
   booleano público isExpDtValid() {
      Calendario cal = Calendario.getInstance();
      cal.setTime(nueva fecha());
      int mm = cal.get(Calendario.MES) + 1; int yy =
      cal.get(Calendario.AÑO); resultado booleano =

         (aa > expAA) || ((aa == expAA) && (mm > expMM)); retorno (!
         resultado);
   }
   booleano privado tiene caracteres válidos () {
      Cadena validChars = "0123456789"; resultado
      booleano = verdadero;
      for (int i = 0; i < cardNum.length(); i++) {
         if (validChars.indexOf(cardNum.substring(i, i + 1)) <
               0) {
            resultado = falso;
            descanso;
         }
      }
      resultado devuelto;
   }
   booleano privado esValidCheckSum() {
      resultado booleano =
      verdadero; int suma = 0;
      multiplicador int = 1;
      int strLen = cardNum.longitud(); for (int i = 0; i <
      strLen; i++) {
         Cadena digit = cardNum.substring(strLen - i - 1,
                                          strLen-i);
```

(continuado)

Listado 38.1 ResumenTarjeta de créditoClase (Continuación)

```
        int currProduct =
          nuevo Entero(dígito).intValue() * multiplicador; si (producto
        actual >= 10)
          suma += (producto actual% 10) + 1; más

          sum += currProduct;
        si (multiplicador == 1)
          multiplicador++;
        más
          multiplicador — ;
      }
      si ((suma% 10) != 0)
        resultado = falso;
      resultado devuelto;
}
/* métodos a ser anulados por subclases. */ public abstract
boolean isNumOfDigitsValid(); público abstracto booleano
isValidPrefix(); public abstract boolean isAccountInGoodStand(); /
* Método final: las subclases no pueden anularse

      * * * MÉTODO DE PLANTILLA***
* /
booleano final público es válido () {
   if (!isExpDtValid()) {
      System.out.println(" Exp. Exp. no válido "); falso retorno;

   }
   if (!esNumOfDigitsValid()) {
      System.out.println("Número de dígitos no válido"); falso retorno;

   }
   si (! es un prefijo válido ()) {
      System.out.println(" Prefijo inválido "); falso retorno;

   }
```

(continuado)

Listado 38.1 ResumenTarjeta de créditoClase (Continuación)

```
        si (! tiene caracteres válidos ()) {
            System.out.println(" Caracteres no válidos "); falso retorno;

        }
        if (!isValidCheckSum()) {
            System.out.println(" Suma de verificación no válida ");
            falso retorno;
        }
        if (!isAccountInGoodStand()) {
            Sistema.fuera.println(
                "La cuenta está inactiva/revocada/por encima del límite"); falso
                retorno;
        }
        devolver verdadero;
    }
}
```

En lenguaje de programación Java,una subclase no puede anular los siguientes dos tipos de métodos de su clase principal:

- métodos privados
- fimétodos finales independientemente del especificador de acceso asociado

Definamos tres subclases:VisaCard, MasterCardyDinersCard - delPlantilla de tarjeta de créditoclase, cada uno proporcionando implementación para todos los métodos abstractos declarados en la clase principal (Listado 38.2 al Listado 38.4).

La asociación de clases resultante se puede representar como enFigura 38.2.

Con el diseño anterior implementado, cualquier cliente que busque validar la información de la tarjeta de crédito simplemente crearía una instancia de unTarjeta de crédito subclase e invocar laes válidamétodo.

```
Cliente de clase pública {
    public static void main(String[] args) {
        Tarjeta de crédito cc =
            nueva VisaCard("1234123412341234,”11, 2004); si
            (cc.isValid())
            System.out.println("Información de tarjeta de crédito válida");
    }
}
```

Listado 38.2Tarjeta VisaClase

```
VisaCard de clase pública extiende la tarjeta de crédito {
  VisaCard pública (String num, int expMonth, int expYear) {
    super(num, expMes, expYear);
  }
  público booleano esNumOfDigitsValid() {
    if ((numerotarjeta.longitud() == 13) ||
        (cardNum.longitud() == 16)) { volver
            verdadero;
    } más {
      devolver falso;
    }
  }
  público booleano esPrefijoVálido() {
    Prefijo de cadena = cardNum.substring(0, 1); si
    (prefijo.equals("4"))          {
      devolver verdadero;
    } más {
      devolver falso;
    }
  }
  public boolean isAccountInGoodStand() {
    /*
      Realice las llamadas necesarias a la API de VISA para
      realizar otras comprobaciones.
    * /
    devolver verdadero;
  }
}
```

NOTAS ADICIONALES

Algoritmo de dígito de control Mod 10

En general, un dígito de control es un dígito agregado a un número que ayuda a verificar la autenticidad del número. El algoritmo de dígito de control Mod 10 se puede utilizar para validar dicho número asociado con un dígito de control.

Listado 38.3tarjeta MasterCardClase

```
clase pública MasterCard extiende CreditCard {
  public MasterCard(String num, int expMonth, int expYear) {
    super(num, expMes, expYear);
  }
  público booleano esNumOfDigitsValid() {
    if (NúmeroTarjeta.longitud() == 16) {
      devolver verdadero;
    } más {
      devolver falso;
    }
  }
  público booleano esPrefijoVálido() {
    Prefijo de cadena = cardNum.substring(0, 1); Cadena
    nextChar = cardNum.substring(1, 2); Cadena validChars =
    "12345";
    //51-55
    si ((prefijo.equals("5")) &&
        (validChars.indexOf(nextChar) >= 0)) { volver
          verdadero;
    } más {
      devolver falso;
    }
  }
  public boolean isAccountInGoodStand() {
    /*
      Realice las llamadas necesarias a la API de MASTER CARD para
      realizar otras comprobaciones.
    * /
    devolver verdadero;
  }
}
```

Listado 38.4DinersCardClase

```
Clase pública DinersCard extiende CreditCard {
  public DinersCard(String num, int expMonth, int expYear) {
    super(num, expMes, expYear);
  }
  público booleano esNumOfDigitsValid() {
    if (NúmeroTarjeta.longitud() == 14) {
      devolver verdadero;
    } más {
      devolver falso;
    }
  }
  público booleano esPrefijoVálido() {
    Prefijo de cadena = cardNum.substring(0, 1); Cadena
    nextChar = cardNum.substring(1, 2); Cadena caracteres
    válidos = "068";
    //51-55
    si ((prefijo.equals("3")) &&
        (validChars.indexOf(nextChar) >= 0)) { volver
          verdadero;
    } más {
      devolver falso;
    }
  }
  public boolean isAccountInGoodStand() {
    /*
      Realice las llamadas necesarias a la API de DINERS CARD para
      realizar otras comprobaciones.
    * /
    devolver verdadero;
  }
}
```

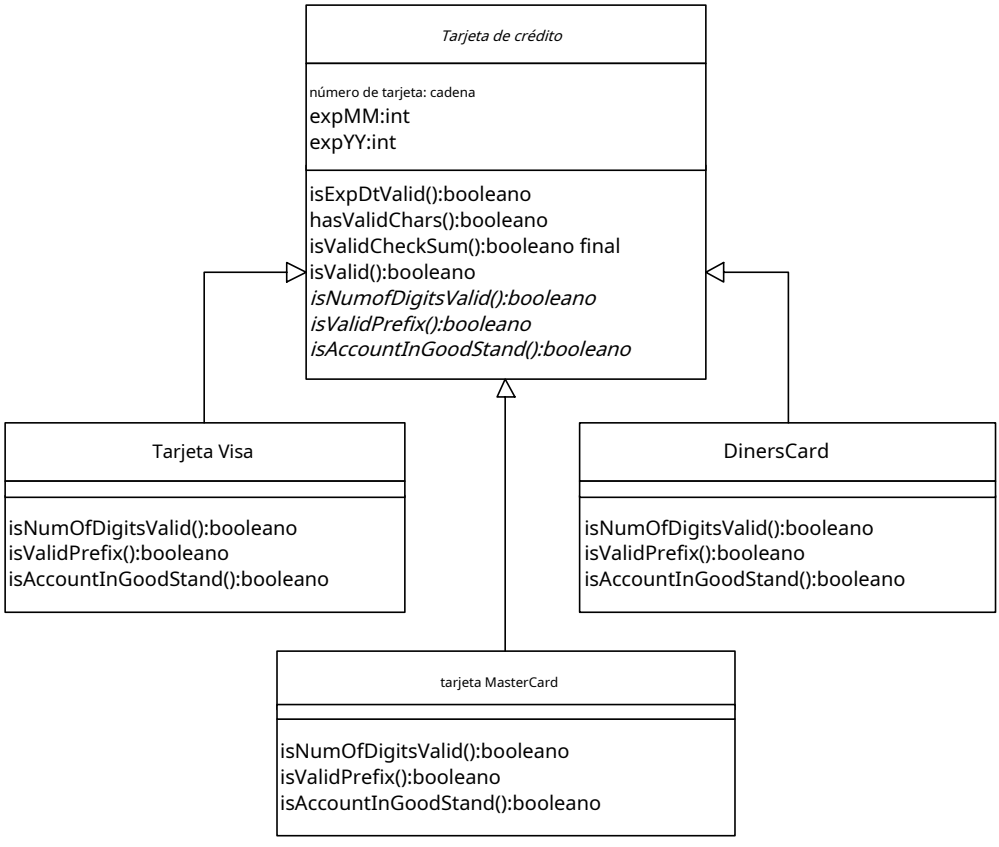

Figura 38.2Tarjeta de créditoJerarquía de clases

Los siguientes pasos describen el proceso de validación:

1. Use 194774915 (dígito de control 5 incluido) como ejemplo. 1 9 4 7 7 4 9 1 5
2. A partir del segundo dígito de la derecha, multiplica cada dígito alternativo por 2. 1 9x2 4 7x2 7 4x2 9 1x2 5

   Resultado:

   1 18 4 14 7 8 9 2 5
3. Agregue dígitos individuales en los productos recién formados. 1 1+8 4 1+4 7 8 9 2 5

   Resultado:

   1 9 4 5 7 8 9 2 5
4. Ahora suma todos los dígitos del número resultante del paso anterior. 1 +9 +4 +5 +7 +8 +9 +2 +5 = 50

5. Ahora divide la suma por 10.

Resultado:

50/10 no deja resto y un resto cero prueba que el número es válido.

PREGUNTAS DE PRÁCTICA

1. Identifique cómo se usa el patrón del método de plantilla cuando diseña un subprograma con código personalizado en cualquiera de los métodos del ciclo de vida del subprograma (iniciar, iniciar, pintar, detener y destruir).
2. Algunos escenarios que involucran muchas implementaciones diferentes para diferentes pasos de un algoritmo podrían conducir a una jerarquía de clases de rápido crecimiento con una gran cantidad de subclases. ¿Qué alternativas consideraría en estos casos?