Patrones en POO

Patrones

Elizabeth León Guzmán, Ph.D.
eleonguz@unal.edu.co

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andres Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Data Mining – Grupo de Investigación en Minería de Datos – (Midas)
Research Group on Artificial Life – Grupo de Investigación en Vida Artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Agenda

Definición de Patrón

Definición

Un patrón corresponde con la identificación y solución a un problema que ocurre de forma repetitiva, de tal forma que se puede usar la misma solución todas las veces que sea necesario.

Ejemplo

Se quiere crear una clase `Caja` que tenga un método `decorar` que tome un entero y lo decore, es decir el método debe devolver una cadena con el entero rodeado de asteriscos. Para un número entero 10 la cadena sería:

```
****
*10*
****
```

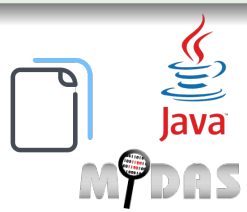
Patrones I

Ejemplo (continuación)

```
public class Caja {
  public String decorar( int n ) {
    String s = ""+n;
    String linea = "*";
    for( int i=0; i<s.length(); i++) {
      linea += "*";
    }
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```

Un programa principal que usa la clase `Caja` puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja();
        String s = caja.decorar(10);
        System.out.println(s);
    }
}
```



Patrones II

Ejemplo

Ahora se quiere crear otro método en la clase `Caja` que tome un `double` y lo decore, es decir devolver una cadena con el double rodeado de asteriscos. Para un `double` 14.56 la cadena sería:

****

14.56

****

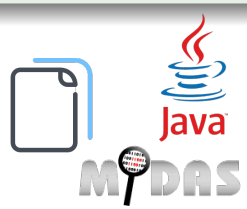
Patrones III

Ejemplo (continuación)

```
public class Caja {
  ...Dejamos el otro método
  public String decorar( double x ) {
    String s = ""+x;
    String linea = "*";
    for( int i=0; i<s.length(); i++) {
      linea += "*";
    }
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```

Un programa principal que usa este método de la clase Caja puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja();
        String s = caja.decorar(-20.34);
        System.out.println(s);
    }
}
```



Patrones IV

Como se puede observar, el código de los dos métodos es casi idéntico, solo cambian el tipo de dato y el nombre de la variable que recibe el método.

```
public String decorar( int n ) {
  String s = ""+n;
  String linea = "*";
  for( int i=0; i<s.length(); i++) {
    linea += "*";
  }
  linea += "*";
  return linea + "\n*" + s + "*\n" + linea;
}
```

```
public String decorar( double x ) {
  String s = ""+x;
  String linea = "*";
  for( int i=0; i<s.length(); i++) {
    linea += "*";
  }
  linea += "*";
  return linea + "\n*" + s + "*\n" + linea;
}
```

Aquí se se puede apreciar que aparece un **patrón** de programación.

Ejemplo

Ahora se quiere crear otro método en la clase `Caja` que tome una cadena de caracteres y la decore, es decir devolver una cadena con la cadena rodeada de asteriscos.
¿Cómo sería el método?

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Patrones V

Ejemplo (continuación)

```
public class Caja {
  ...Dejamos los otros métodos
  public String decorar( String s ) {
    String linea = "*";
    for( int i=0; i<s.length(); i++) {
      linea += "*";
    }
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```

Un programa principal que usa este método de la clase `Caja` puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja();
        String s = caja.decorar("Pedro Páramo");
        System.out.println(s);
    }
}
```

Patrones - Abstracción I

De nuevo, el código es muy similar al anterior de decorar un `int` o un `double`, es decir, tenemos un patrón de programación.

Ahora podemos reducir estos tres métodos a un solo método aprovechándonos de las propiedades de herencia y amoldamiento de datos: los tipos básicos `int` y `double` se amoldan a `Integer` y `Double`, respectivamente y estos heredan de `Object`.

MIDAS

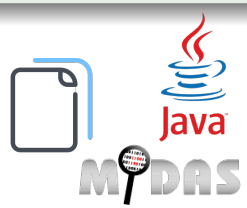
Patrones - Abstracción II

Ejemplo (continuación)

```
public class Caja {
  // Eliminamos los otros métodos
  public String decorar( Object obj ) {
    String s = obj.toString();
    String linea = "*";
    for( int i=0; i<s.length(); i++) {
      linea += "*";
    }
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```

Un programa principal que usa este método de la clase Caja puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja();
        System.out.println( caja.decorar("Pedro Páramo") );
        System.out.println( caja.decorar(10.34) );
    }
}
```



Agenda

Patrones y Herencia I

Patrones como el anterior se pueden hacer mucho más genéricos que el obtenido de simplemente llevar todo a la clase `Object`, pues esto 'pierde' propiedades.

Ejemplo

Se quiere que la clase `Caja` mantenga como atributo el objeto que se quiere decorar, por lo tanto, el método `decorar` ya no lo recibe como argumento, sino el constructor. Adicionalmente se quiere un método `obtener` que retorne el objeto que se decora:

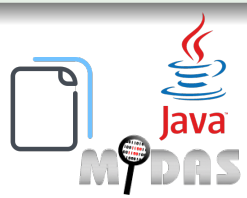
Patrones y Herencia II

Ejemplo (continuación)

```
public class Caja {
  protected Object obj;
  public Caja(Object obj){  this.obj = obj; }
  public Object obtener(){ return obj; }
  public String decorar() {
    String s = obj.toString();
    String linea = "*";
    for( int i=0; i<s.length(); i++) linea += "*";
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```

Un programa principal que usa métodos de la clase Caja puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja("Pedro Páramo");
        System.out.println( caja.decorar() );
        System.out.println( caja.obtener() );
    }
}
```



Patrones y Herencia - Limitaciones

En el ejemplo anterior, aunque se sabe que lo que se guardo es una cadena de caracteres, el método obtener no puede decirnos que es una cadena pues simplemente sabe que es un `Object`, perdió ese conocimiento, por eso es que el siguiente código genera un error de compilación:

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja("Pedro Páramo");
        char c = caja.obtener().charAt(2);
    }
}
```


```
Multiple markers at this line
        - The method charAt(int) is undefined for the type
          Object
        - Syntax error on token "=", ; expected
```

Patrones, Herencia y Amoldamiento

Una forma de corregir el error es amoldar el objeto retornado al requerido, si sabemos su tipo (aunque se hace un poco ilegible el mismo):

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        Caja caja = new Caja("Pedro Páramo");
        char c = ((String)caja.obtener()).charAt(2);
    }
}
```

Otra forma es usando genericidad.

Agenda

Definición de Genericidad

Definición

Una interfaz o clase genérica permite parametrizar los tipos de sus atributos y los argumentos y/o valores de retorno de sus métodos para conservar las propiedades inherentes de dichos tipos de datos.

Para definir una clase o interfaz genérica se utiliza el operador diamante <>.

Vamos a volver genérica la clase `Caja`

Generecidad I

Ejemplo (continuación)

```
public class Caja<T> {
  protected T obj;
  public Caja(T obj){  this.obj = obj; }
  public T obtener(){ return obj; }
  public String decorar() {
    String s = obj.toString();
    String linea = "*";
    for( int i=0; i<s.length(); i++)
      linea += "*";
    linea += "*";
    return linea + "\n*" + s + "*\n" + linea;
  }
}
```


En el nombre de la clase se usa el operador diamante con los tipos parametrizados. Se parametriza el atributo `obj` de la clase `Caja` y se dice que el tipo es `T`. Notemos como el atributo, el constructor y el método `obtener` se ajustan a ese tipo parametrizado.

Generecidad II

Ejemplo (continuación)

Un programa principal que usa métodos de la clase `Caja` puede ser:

```
public class Main {
    public static void main(String[] args) {
        Caja<String> caja = new Caja<String>("Pedro Páramo");
        System.out.println( caja.decorar() );
        char c = caja.obtener().charAt(2);
        System.out.println(c);
        Caja<Double> caja2 = new Caja<Double>(20.34);
        System.out.println( caja2.decorar() );
        double y = caja2.obtener() + 20.0;
        System.out.println(y);
    }
}
```

Se definen dos cajas una que almacena una cadena y otra que almacena un real. Se usa el operador diamante en los constructores e instancias de objetos. De esta manera se pueden usar las propiedades de dichos tipos en el programa.

Generecidad III

Ejemplo

```
public class Pareja<K,V> {
  protected K clave;
  protected V valor;
  public Pareja(K clave, V valor){
    this.clave = clave;
    this.valor = valor;
  }
  public K clave(){ return clave; }
  public V valor(){ return valor; }
  public String toString(){
    return "("+clave+","+valor+")";
  }
}
```


Un ejemplo de una clase con dos tipos parametrizados. Aquí se parametriza el tipo del atributo `clave` que es el primer parámetro (`K`) y el atributo `valor` que es el segundo parámetro (`V`) del operador diamante.

Generecidad IV

Ejemplo (continuación)

```
public class Main {
    public static void main(String[] args) {
        Pareja<Integer,String> a =
          new Pareja<Integer,String>(20,"Pedro Páramo");
        System.out.println( pareja );
        Integer c = pareja.clave() + 10;
        System.out.println( c );
        System.out.println(pareja.valor().charAt(2));
    }
}
```


Un ejemplo de una programa principal con la clase genérica de dos parámetros `Pareja`.

Agenda

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Conectando todo I

Vamos a conectar todos los elementos de programación orientada por objetos en un ejemplo de optimización simple, el ascenso a la colina.

Ejemplo

Se quiere realizar un programa orientado por objetos que permita optimizar una función asociada a un tipo de dato particular (el cual puede ser un real, entero, arreglo de binarios o cadena de caracteres) y que devuelve un real, usando el método del ascenso a la colina.
Código disponible en el repositorio:
https://github.com/jgomezpe/ascensocolina

Conectando todo II

Ejemplo (continuación)

Definimos una interfaz genérica para modelar la función que se está optimizando:

```
public interface Funcion<T> {
   double evaluar(T x);
}
```

Notemos que es una interfaz que parametriza el tipo de dato que recibe la función a evaluar (tipo genérico `T`)

Conectando todo III

Ejemplo (continuación)

Ahora definimos un par de funciones para enteros:

```
public class Primera implements Funcion<Integer> {
    @Override
    public double evaluar(Integer x){
      return -x*x;
    }
}
```

```
public class Segunda implements Funcion<Integer> {
    @Override
    public double evaluar(Integer x){
      return 2*x;
    }
}
```

Notemos que estas clases definen directamente el tipo en el operador diamante.

MIDAS

Conectando todo IV

Ejemplo (continuación)

Ahora definimos un par de funciones para reales:

```
public class Tercera implements Funcion<Double> {
    @Override
    public double evaluar(Double x){
      return Math.abs(x);
    }
}
```

```
public class Cuarta implements Funcion<Double> {
    @Override
    public double evaluar(Double x){
      return 3*x*x+2*x-1;
    }
}
```

Notemos que estas clases definen directamente el tipo en el operador diamante.

Conectando todo V

Ejemplo (continuación)

Ahora definimos una función para cadenas:

```
public class SoloA implements Funcion<String> {
    @Override
    public double evaluar(String x){
      int cuenta=0;
      for( int i=0; i<x.length(); i++ ){
        if(x.charAt(i)=='a' || x.charAt(i)=='A')
         cuenta++;
      }
      return cuenta;
    }
}
```

Notemos que esta clase define directamente el tipo en el operador diamante.

MIDAS

Conectando todo VI

Ejemplo (continuación)

Definimos una interfaz genérica para modelar la mutación:

```
public interface Mutacion<T> {
    T aplicar(T x);
}
```

Notemos que es una interfaz que parametriza el tipo de dato que recibe la función aplicar (tipo genérico T)

Conectando todo VII

Ejemplo (continuación)

Ahora definimos mutaciones para los enteros:

```
public class MutacionEntero implements Mutacion<Integer> {
    @Override
    public Integer aplicar(Integer x){
      x += (int)(Math.random()*5);
      if( x >= 10 ) x = 10;
      else if( x<-10) x=-10;
      return x;
    }
}
```

Notemos que esta clase define directamente el tipo en el operador diamante.

Conectando todo VIII

Ejemplo (continuación)

Ahora definimos mutaciones para los reales:

```
public class MutacionReal implements Mutacion<Double> {
    @Override
    public Double aplicar(Double x){
      x += Math.random();
      if( x >= 5.0 ) x = 5.0;
      else if( x<-5.0 ) x=-5.0;
      return x;
    }
}
```

Notemos que esta clase define directamente el tipo en el operador diamante.

Conectando todo IX

Ejemplo (continuación)

Ahora definimos mutaciones para las cadenas:

```
public class MutacionCadena implements Mutacion<String> {
    @Override
    public String aplicar(String x){
      char c = (char)(65+(int)(Math.random()*26));
      int pos = (int)(Math.random()*x.length());
      return x.substring(0,pos)+c+x.substring(pos,x.length());
    }
}
```

Notemos que esta clase define directamente el tipo en el operador diamante.

Conectando todo X

Ejemplo (continuación)

Ahora definimos el ascenso a la colina:

```
public class AscensoColina<T>{
  public T aplicar( Funcion<T> f, T x, Mutacion<T> m, int ITERS ){
    double fx = f.evaluar(x);
    for( int i=0; i<ITERS; i++ ){
      T y = m.aplicar(x);
      double fy = f.evaluar(y);
      if( fy >= fx ){
        x = y;
        fx = fy;
      }
      System.out.println("x="+x+" fx="+fx);
    }
    return x;
  }
}
```

Conectando todo XI

Ejemplo (continuación)

Finalmente, el programa principal (para cadenas):

```
public class Main{
  public static void main(String[] args){
    AscensoColina<String> opt = new AscensoColina<String>();
    Funcion<String> f = new SoloA();
    Mutacion<String> m = new MutacionCadena();
    opt.aplicar(f, "A ver que pasa", m, 1000);
  }
}
```

MIDAS

Conectando todo XII

Ejemplo (continuación)

Finalmente, el programa principal (para reales):

```
public class Main{
  public static void main(String[] args){
    AscensoColina<Double> opt = new AscensoColina<Double>();
    Funcion<Double> f = new Cuarta();
    // Funcion<Double> f = new Tercera();
    Mutacion<Double> m = new MutacionReal();
    opt.aplicar(f, 1.0, m, 1000);
  }
}
```

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un