5.4. Patrones de Algoritmo para Instrucciones Repetitivas

Cuando trabajamos con estructuras contenedoras, las soluciones de muchos de los problemas que debemos resolver son similares y obedecen a ciertos esquemas ya conocidos (¿cuántas personas no habrán resuelto ya los mismos problemas que estamos aquí resolviendo?). En esta sección pretendemos identificar tres de los patrones que más se repiten en el momento de escribir un ciclo, y con los cuales se pueden resolver todos los problemas del caso de estudio planteados hasta ahora. Lo ideal sería que, al leer un problema que debemos resolver (el método que debemos escribir), pudiéramos identificar el patrón al cual corresponde y utilizar las guías que existen para resolverlo. Eso simplificaría enormemente la tarea de escribir los métodos que tienen ciclos.

Un patrón de algoritmo se puede ver como una solución genérica para un tipo de problemas, en la cual el programador sólo debe resolver los detalles particulares de su problema específico.

En esta sección vamos a introducir tres patrones que se diferencian por el tipo de recorrido que hacemos sobre la secuencia.

5.4.1. Patrón de Recorrido Total

En muchas ocasiones, para resolver un problema que involucra una secuencia, necesitamos recorrer todos los elementos que ésta contiene para lograr la solución. En el caso de estudio de las notas tenemos varios ejemplos de esto:

- Calcular la suma de todas las notas.
- Contar cuántos en el curso obtuvieron la nota 3,5.
- Contar cuántos estudiantes aprobaron el curso.
- Contar cuántos en el curso están por debajo del promedio (conociendo este valor).
- Aumentar en 10% todas las notas inferiores a 2,0.

¿Qué tienen en común los algoritmos que resuelven esos problemas? La respuesta es que la solución requiere siempre un recorrido de todo el arreglo para poder cumplir el objetivo que se está buscando: debemos pasar una vez por cada una de las casillas del arreglo. Esto significa:

1. Que el índice para iniciar el ciclo debe empezar en cero.
2. Que la condición para continuar es que el índice sea menor que la longitud del arreglo.
3. Que el avance consiste en sumarle uno al índice.

Esa estructura que se repite en todos los algoritmos que necesitan un recorrido total es lo que denominamos el **esqueleto del patrón**, el cual se puede resumir con el siguiente fragmento de código:

```
for( int indice = 0; indice < arreglo.length; indice++ )
{
  <cuerpo>
}
```

- Es común que en lugar de la variable " `indice` " se utilice una variable llamada " `i` ". Esto hace el código un poco más compacto.
- En lugar del operador " `length` ", se puede utilizar también la constante que indica el número de elementos del arreglo.
- Los corchetes del " `for` " sólo son necesarios si el cuerpo tiene más de una instrucción.

Lo que cambia en cada caso es lo que se quiere hacer en el cuerpo del ciclo. Aquí hay dos variantes principales. En la primera, algunos de los elementos del arreglo van a ser modificados siguiendo una regla (por ejemplo, aumentar en 10% todas las notas inferiores a 2,0). Lo único que se hace en ese caso es reemplazar el del esqueleto por las instrucciones que hacen la modificación pedida a un elemento del arreglo (el que se encuentra en la posición indice). Esa variante se ilustra en el ejemplo 4.

Ejemplo 4

Objetivo: Mostrar la primera variante del patrón de recorrido total.

En este ejemplo se presenta la implementación del método de la clase Curso que aumenta en 10% todas las notas inferiores a 2,0.

```
public void hacerCurva( )
{
  for( int i = 0; i < notas.length; i++ )
  {
    if( notas[ i ] < 2.0 )
      notas[ i ] = notas[ i ] * 1.1;
  }
}
```

- El esqueleto del patrón de algoritmo de recorrido total se copia dentro del cuerpo del método.
- Se reemplaza el cuerpo del patrón por la instrucción condicional que hace la modificación pedida.
- En el cuerpo se indica la modificación que debe sufrir el elemento que está siendo

referenciado por el índice con el que se recorre el arreglo.

La segunda variante corresponde a calcular alguna propiedad sobre el conjunto de elementos del arreglo (por ejemplo, contar cuántos estudiantes aprobaron el curso). Esta variante implica cuatro decisiones que definen la manera de completar el esqueleto del patrón:

1. Cómo acumular la información que se va llevando a medida que avanza el ciclo.
2. Cómo inicializar dicha información.
3. Cuál es la condición para modificar dicho acumulado en el punto actual del ciclo.
4. Cómo modificar el acumulado.

En el ejemplo 5 se ilustra esta variante.

Ejemplo 5

Objetivo: Mostrar la segunda variante del patrón de recorrido total.

En este ejemplo se presenta la aplicación del patrón de algoritmo de recorrido total, para el problema de contar el número de estudiantes que aprobaron el curso.

- ¿Cómo acumular información?

Vamos a utilizar una variable de tipo entero llamada `vanAprobando`, que va llevando durante el ciclo el número de estudiantes que aprobaron el curso.

- ¿Cómo inicializar el acumulado?

La variable `vanAprobando` se debe inicializar en 0, puesto que inicialmente no hemos encontrado todavía ningún estudiante que haya pasado el curso.

- ¿Condición para cambiar el acumulado?

Cuando `notas[ indice ]` sea mayor o igual a 3,0, porque quiere decir que hemos encontrado otro estudiante que pasó el curso.

- ¿Cómo modificar el acumulado?

El acumulado se modifica incrementándolo en 1.

```
public int cuantosAprobaron( )
{

  int vanAprobando = 0;

  for( int i = 0; i < notas.length; i++ )
  {
    if( notas[ i ] >= 3.0 ) vanAprobando++;
  }

  return vanAprobando;
}
```

- Las cuatro decisiones tomadas anteriormente van a definir la manera de completar el esqueleto del algoritmo definido por el patrón.
- Las decisiones 1 y 2 definen el inicio del ciclo.
- Las decisiones 3 y 4 ayudan a construir el cuerpo del mismo.

> En resumen, si el problema planteado corresponde al patrón de recorrido total, se debe identificar la variante y luego tomar las decisiones que definen la manera de completar el esqueleto.

Tarea 3

Objetivo: Generar habilidad en el uso del patrón de algoritmo de recorrido total.

Escriba los métodos de la clase Curso que resuelven los siguientes problemas, los cuales corresponden a las dos variantes del patrón de algoritmo de recorrido total.

Escriba un método para modificar las notas de los estudiantes de la siguiente manera: a todos los que obtuvieron más de 4,0, les quita 0,5. A todos los que obtuvieron menos de 2,0, les aumenta 0,5. A todos los demás, les deja la nota sin modificar:

```
public void cambiarNotas( )
{









}
```

Escriba un método que retorne la menor nota del curso:

```
public double menorNota ( )
{









}
```

Escriba un método que indique en qué rango hay más notas en el curso: rango 1 de 0,0 a 1,99, rango 2 de 2,0 a 3,49, rango 3 de 3,5 a 5,0:

```
public int rangoConMasNotas( )
{









}
```

5.4.2. Patrón de Recorrido Parcial

En algunos problemas de manejo de secuencias no es necesario recorrer todos los elementos para lograr el objetivo propuesto. Piense en la solución de los siguientes problemas:

- Informar si algún estudiante obtuvo la nota 5,0.
- Buscar el primer estudiante con nota igual a cero.
- Indicar si más de 3 estudiantes perdieron el curso.
- Aumentar el 10% en la nota del primer estudiante que haya sacado más de 4,0.

En todos esos casos hacemos un recorrido del arreglo, pero éste debe terminar tan pronto hayamos resuelto el problema. Por ejemplo, el método que informa si algún estudiante obtuvo cinco en la nota del curso debe salir del proceso iterativo tan pronto localice el

primer estudiante con esa nota. Sólo si no lo encuentra, va a llegar hasta el final de la secuencia.

Un **recorrido parcial** se caracteriza porque existe una condición que debemos verificar en cada iteración para saber si debemos detener el ciclo o volver a repetirlo.

En este patrón, debemos adaptar el esqueleto del patrón anterior para que tenga en cuenta la condición de salida, de la siguiente manera:

```
boolean termino = false;

for( int i = 0; i < arreglo.length && !termino; i++ )
{
  <cuerpo>

  if( <ya se cumplió el objetivo> )
    termino = true;
}
```

- Primero, declaramos una variable de tipo `boolean` para controlar la salida del ciclo, y la inicializamos en false.
- Segundo, en la condición del ciclo usamos el valor de la variable que acabamos de definir: si su valor es verdadero, no debe volver a iterar.
- Tercero, en algún punto del ciclo verificamos si el problema ya ha sido resuelto (si ya se cumplió el objetivo). Si ése es el caso, cambiamos el valor de la variable a verdadero.

```
for( int i = 0; i < arreglo.length && !<condición>; i++ )
{
  <cuerpo>
}
```

Este patrón de esqueleto es más simple que el anterior, pero sólo se debe usar si la expresión que indica que ya se cumplió el objetivo del ciclo es sencilla.

> Cuando se aplica el patrón de recorrido parcial, el primer paso que se debe seguir es identificar la condición que indica que el problema ya fue resuelto. Con esa información se puede tomar la decisión de cuál esqueleto de algoritmo es mejor usar.

Ejemplo 6

Objetivo: Mostrar el uso del patrón de recorrido parcial para resolver un problema.

En este ejemplo se presentan tres soluciones posibles al problema de decidir si algún estudiante obtuvo cinco en la nota del curso.

```
public boolean alguienConCinco( )
{
  boolean termino = false;

  for( int i = 0; i < notas.length && !termino; i++ )
  {
    if( notas[ i ] == 5.0 )
    termino = true;
  }

  return termino;
}
```

- La condición para no seguir iterando es que se encuentre una nota igual a 5,0 en la posición `i`.
- Al final del método, se retorna el valor de la variable "`termino`", que indica si el objetivo se cumplió. Esto funciona en este caso particular, porque dicha variable dice que en el arreglo se encontró una nota igual al valor buscado.

```
public boolean alguienConCinco( )
{
  int i = 0;
  while( i < notas.length && notas[ i ] != 5.0 )
  {
    i++;
  }
  return i < notas.length;
}
```

- Esta es la segunda solución posible, y evita el uso de la variable "`termino`", pero tiene varias consecuencias sobre la instrucción iterativa.
- En lugar de la instrucción `for` es más conveniente usar la instrucción `while`.
- La condición de continuación en el ciclo es que la i-ésima nota sea diferente de 5,0.
- El método debe retornar verdadero si la variable `i` no llegó hasta el final del arreglo, porque esto querría decir que encontró en dicha posición una nota igual a cinco.

```
public boolean alguienConCinco( )
{
  for( int i = 0; i < notas.length; i++ )
  {
    if( notas[ i ] == 5.0 )
      return true;
  }
  return false;
}
```

- Esta es la tercera solución posible. Si dentro del ciclo ya tenemos la respuesta del método, en lugar de utilizar la condición para salir del ciclo, la usamos para salir de todo el método.
- En la última instrucción retorna falso, porque si llega a ese punto quiere decir que no encontró ninguna nota con el valor buscado.
- Esta manera de salir de un ciclo, terminando la ejecución del método en el que éste se encuentra, se debe usar con algún cuidado, puesto que se puede producir código difícil de entender.

> Hay muchas soluciones posibles para resolver un problema. Un patrón de algoritmo sólo es una guía que se debe adaptar al problema específico y al estilo preferido del programador.

Para el patrón de recorrido parcial aparecen las mismas dos variantes que para el patrón de recorrido total (ver ejemplo 7):

- En la primera variante se modifican los elementos del arreglo hasta que una condición se cumpla (por ejemplo, encontrar las tres primeras notas con 1,5 y asignarles 2,5). En ese caso, en el cuerpo del método va la modificación que hay que hacerle al elemento que se encuentra en el índice actual, pero se debe controlar que cuando haya llegado a la tercera modificación termine el ciclo.
- En la segunda variante, se deben tomar las mismas cuatro decisiones que se tomaban con el patrón de recorrido total, respecto de la manera de acumular la información para calcular la respuesta que está buscando el método.

Ejemplo 7

Objetivo: Mostrar el uso del patrón de recorrido parcial, en sus dos variantes.

En este ejemplo se presentan dos métodos de la clase Curso, en cada uno de los cuales se ilustra una de las variantes del patrón de recorrido parcial.

Encontrar las primeras tres notas iguales a 1,5 y asignarles 2,5:

```
public void subirNotas( )
{
  int numNotas  =  0;
  for( int i =  0; i <  notas.length &&  numNotas  <  3; i++ )
  {
    if( notas[ i ] == 1,5 )
    {
      numNotas++;
      notas[ i ] =  2,5;
    }
  }
}
```

Este método corresponde a la primera variante, porque hace una modificación de los elementos del arreglo hasta que una condición se cumpla. En el método del ejemplo, debemos contar el número de modificaciones que hacemos, para detenernos al llegar a la tercera.

Retornar la posición en la secuencia de la tercera nota con valor 5,0. Si dicha nota no aparece al menos 3 veces, el método debe retornar el valor –1:

```
public int tercerCinco( )
{
  int cuantosCincos = 0;
  int posicion = -1;
  for( int i =  0; i < notas.length && posicion == -1; i++ )
  {
    if( notas[ i ] == 5,0 )
    {
      cuantosCincos++;
      if( cuantosCincos == 3 )
      {
        posicion = i;
      }
    }
  }
  return posicion;
}
```

- ¿Cómo acumular información? En este caso necesitamos dos variables para acumular la información: la primera para llevar el número de notas iguales a 5,0 que han aparecido (`cuantosCincos`), la segunda para indicar la posición de la tercera nota 5,0 (`posicion`).
- ¿Cómo inicializar el acumulado? La variable `cuantosCincos` debe comenzar en 0. La variable `posicion` debe comenzar en menos 1.
- ¿Condición para cambiar el acumulado? Si la nota actual es 5,0 debemos cambiar

nuestro acumulado.

- ¿Cómo modificar el acumulado? Debe cambiar la variable `cuantosCincos`, incrementándose en 1. Si es el tercer 5,0 de la secuencia, la variable `posicion` debe cambiar su valor, tomando el valor del índice actual.

Tarea 4

Objetivo: Generar habilidad en el uso del patrón de algoritmo de recorrido parcial.

Escriba los métodos de la clase Curso que resuelven los siguientes problemas, los cuales corresponden a las dos variantes del patrón de algoritmo de recorrido parcial.

Reemplazar todas las notas del curso por 0,0, hasta que aparezca la primera nota superior a 3,0.

```
public void notasACero( )
{









}
```

Calcular el número mínimo de notas del curso necesarias para que la suma supere el valor 30, recorriéndolas desde la posición 0 en adelante. Si al sumar todas las notas no se llega a ese valor, el método debe retornar –1.

```
public int sumadasDanTreinta( )
{









}
```

5.4.3. Patrón de Doble Recorrido

El último de los patrones que vamos a ver en este capítulo es el de doble recorrido. Este patrón se utiliza como solución de aquellos problemas en los cuales, por cada elemento de la secuencia, se debe hacer un recorrido completo. Piense en el problema de encontrar la nota que aparece un mayor número de veces en el curso. La solución evidente es tomar la primera nota y hacer un recorrido completo del arreglo contando el número de veces que ésta vuelve a aparecer. Luego, haríamos lo mismo con los demás elementos del arreglo y escogeríamos al final aquélla que aparezca un mayor número de veces.

El esqueleto básico del algoritmo con el que se resuelven los problemas que siguen este patrón es el siguiente:

```
for( int indice1 = 0; indice1 < arreglo.length; indice1++ )
{
  for( int indice2 = 0; indice2 < arreglo.length; indice2++ )
  {
    <cuerpo del ciclo interno>
  }

  <cuerpo del ciclo externo>
}
```

- El ciclo de afuera está controlado por la variable " `indice1` ", mientras que el ciclo interno utiliza la variable " `indice2` ".
- Dentro del cuerpo del ciclo interno se puede hacer referencia a la variable " `indice1` ".

Las variantes y las decisiones son las mismas que identificamos en los patrones anteriores. La estrategia de solución consiste en considerar el problema como dos problemas independientes, y aplicar los patrones antes vistos, tal como se muestra en el ejemplo 8.

Ejemplo 8

Objetivo: Mostrar el uso del patrón de algoritmo de recorrido total.

En este ejemplo se muestra el método de la clase Curso que retorna la nota que aparece un mayor número de veces. Para escribirlo procederemos por etapas, las cuales se describen en la parte derecha.

```
public double masVecesAparece( )
{

  double notaMasVecesAparece = 0.0;

  for( int i = 0; i < notas.length; i++ )
  {

    for( int j = 0; j < notas.length; j++ )
    {

      //Por completar

    }
  }

  return notaMasVecesAparece;
}
```

- Primera etapa: armar la estructura del método a partir del esqueleto del patrón.
- Utilizamos las variables `i` y `j` para llevar los índices en cada uno de los ciclos.
- Decidimos que el resultado lo vamos a dejar en una variable llamada `notasMasVecesAparece`, la cual retornamos al final del método.
- Una vez construida la base del método, identificamos los dos problemas que debemos resolver en su interior: (1) contar el número de veces que aparece en el arreglo el valor que está en la casilla i; (2) encontrar el mayor valor entre los que son calculados por el primer problema.

```
public double masVecesAparece( )
{
  double notaMasVecesAparece = 0.0;

  for( int i = 0; i < notas.length; i++ )
  {
    double notaBuscada = notas[ i ]; int contador = 0;

    for( int j = 0; j < notas.length; j++ )
    {
      if( notas[ j ] == notaBuscada )
        contador++;
    }

    //Por completar
  }
  return notaMasVecesAparece;
}
```

- Segunda etapa: Resolvemos el primero de los problemas identificados, usando para

eso el ciclo interno.

- Para facilitar el trabajo, vamos a dejar en la variable `notaBuscada` la nota para la cual queremos contar el número de ocurrencias. Dicha variable la inicializamos con la nota de la casilla i.
- Usamos una segunda variable llamada `contador` para acumular allí el número de veces que aparezca el valor buscado dentro del arreglo. Dicho valor será incrementado cuando `notaBuscada == notas[ j ]`.
- Al final del ciclo, en la variable contador quedará el número de veces que el valor de la casilla i aparece en todo el arreglo.

```
public double masVecesAparece( )
{

  double notaMasVecesAparece = 0.0;

  int numeroVecesAparece = 0;

  for( int i = 0; i < notas.length; i++ )
  {
    double notaBuscada = notas[ i ];
    int contador = 0;

    for( int j = 0; j < notas.length; j++ )
    {
      if( notas[ j ] == notaBuscada )
        contador++;
    }

    if( contador > numeroVecesAparece )
    {
      notaMasVecesAparece = notaBuscada;
      numeroVecesAparece = contador;
    }
  }

  return notaMasVecesAparece;

}
```

- Tercera etapa: Usamos el ciclo externo para encontrar la nota que más veces aparece.
- Usamos para eso dos variables: `notaMasVecesAparece` que indica la nota que hasta el momento más veces aparece, y `numeroVecesAparece` para saber cuántas veces aparece dicha nota.
- Luego definimos el caso en el cual debemos cambiar el acumulado: si encontramos un valor que aparezca más veces que el que teníamos hasta el momento (`contador > numeroVecesAparece`) debemos actualizar los valores de nuestras variables.

En general, este patrón dice que para resolver un problema que implique un doble recorrido, primero debemos identificar los dos problemas que queremos resolver (uno con cada ciclo) y, luego, debemos tratar de resolverlos independientemente, usando los patrones de recorrido total o parcial.

Si para resolver un problema se necesita un tercer ciclo anidado, debemos escribir métodos separados que ayuden a resolver cada problema individualmente, tal como se plantea en el nivel 4, porque la solución directa es muy compleja y propensa a errores.

Tarea 5

Objetivo: Generar habilidad en el uso del patrón de algoritmo de doble recorrido.

Escriba el método de la clase Curso que resuelve el siguiente problema, que corresponde al patrón de algoritmo de doble recorrido.

Calcular una nota del curso (si hay varias que lo cumplan puede retornar cualquiera) tal que la mitad de las notas sean menores o iguales a ella.

```
public double notaMediana( )
{









}
```