9.A. Introducción a las estructuras de almacenamiento.

2. Clases y métodos genéricos (I).

Una particularidad esencial de las estructuras dinámicas es la utilización de lo que se conoce como clases y métodos genéricos.

¿Sabes por qué se suele aprender el uso de los genéricos? Pues porque se necesita para usar las listas, aunque realmente los genéricos son una herramienta muy potente y que nos puede ahorrar tareas de programación repetitivas.

Imagen procedente de curso Progragamación del MECD

Las clases y los métodos genéricos son un recurso de programación disponible en muchos lenguajes de programación. Su objetivo es claro: facilitar la reutilización del software, creando métodos y clases que puedan trabajar con diferentes tipos de objetos, evitando incomodas y engorrosas conversiones de tipos. Su inicio se remonta a las plantillas (templates) de C++, un gran avance en el mundo de programación sin duda. En lenguajes de más alto nivel como Java o C# se ha transformado en lo que se denomina "genéricos". Veamos un ejemplo sencillo de como transformar un método normal en genérico:

Versiones genérica y no genérica del método de compararTamano.

Versión no genérica

```
public class util {

   public static int compararTamano(Object[] a,Object[] b) {

      return a.length-b.length;

   }

}
```

Versión genérica del método

```
public class util {

   public static <T> int compararTamano (T[] a, T[] b) {

      return a.length-b.length;

   }

}
```

Los dos métodos anteriores tienen un claro objetivo: permitir comprobar si un array es mayor que otro. Retornarán 0 si ambos arrays son iguales, un número mayor de cero si el array `b` es mayor, y un número menor de cero si el array `a` es mayor, pero uno es genérico y el otro no. La versión genérica del módulo incluye la expresión "`<T>`", justo antes del tipo retornado por el método. "`<T>`" es la definición de una **variable o parámetro formal de tipo** de la clase o método genérico, al que podemos llamar simplemente **parámetro de tipo o parámetro genérico**. Este parámetro genérico (`T`) se puede usar a lo largo de todo el método o clase, dependiendo del ámbito de definición, y hará referencia a cualquier clase con la que nuestro algoritmo tenga que trabajar. Como veremos más adelante, puede haber más de un parámetro genérico.

Utilizar genéricos tiene claras ventajas. Para invocar un método genérico, sólo hay que realizar una **invocación de tipo genérico**, olvidándonos de las conversiones de tipo. Esto consiste en indicar qué clases o interfaces concretas se utilizarán en lugar de cada parámetro genérico ("`<T>`"), para después, pasándole los argumentos correspondientes, ejecutar el algoritmo. Cada clase o interfaz concreta, la podemos denominar **tipo** o **tipo base** y se da por sentado que **los argumentos pasados al método genérico serán también de dicho tipo base**.

Supongamos que el tipo base es `Integer`, pues para realizar la invocación del método genérico anterior basta con indicar el tipo, entre los símbolos de menor qué y mayor qué ("`<Integer>`"), justo antes del nombre del método.

Invocaciones de las versiones genéricas y no genéricas de un método.

Invocación del método no genérico.	Invocación del método genérico.

`Integer []a={0,1,2,3,4};`
`Integer []b={0,1,2,3,4,5};`
`util.compararTamano ((Object[])a, (Object[])b);`
	`Integer []a={0,1,2,3,4};`
	`Integer []b={0,1,2,3,4,5};`
	`util.<Integer>compararTamano (a, b);`