Anexo I.- Generics

Los *Generics* son un concepto introducido en la versión 5 de Java que permiten implementar algoritmos con tipos de datos genéricos, es decir, sin necesidad de indicar en tiempo de diseño el tipo de datos con el que trabajan nuestras clases.
Son importantes puesto que permiten al compilador informar de muchos errores de compilación y permiten reducir drásticamente la cantidad de operaciones de casting , aumentando de esta forma la legibilidad de nuestro código.
Los *Generics* permiten utilizar tipos para parametrizar las clases, interfaces y métodos al definirlos. Los beneficios de utilizar *Generics* son:

- Comprobación de tipo más fuerte en tiempo de compilación.
- Eliminación de castings aumentando la legibilidad del código.
- Posibilidad de implementar algoritmos genéricos con tipado seguro.

Una clase que hace uso de Generics tiene un aspecto similar al siguiente:

```
public class Pila<T> {
    private ArrayList<T> lista;
    public Pila() {
        lista = new ArrayList<>();
    }
    public T push(T e) {
        lista.add(e);
        return e;
    }
    public T pop() {
        T e = top();
        lista.remove(lista.size()-1);
        return e;
    }
    public int size() {
        return lista.size();
    }
    public T top() {
        return lista.get(lista.size()-1);
    }
    public boolean isEmpty() {
        return lista.isEmpty();
    }
    public String toString() {
        return lista.toString();
    }

}
```

Cómo podemos observar donde debe de aparecer el tipo de datos con el que trabaja la Pila, aparece una letra T. El nombre de la letra realmente no tiene por qué ser T, puede ser cualquier cadena de caracteres que no sea una palabra reservada, pero por convenio suelen utilizarse las siguientes letras:

- E: elemento de una colección.
- K: clave

- N: número
- T: tipo
- V: valor
- S, U, V, etc: para segundos, terceros, cuartos tipos, etc.

Una clase genérica puede tener múltiples argumentos de tipos y sus argumentos pueden ser a su vez tipos genéricos. Después del nombre de la clase se puede indicar la lista de parámetros de tipos con el formato <T1, T2, T3, ...Tn> donde T_i como hemos comentado puede ser cualquier cadena de caracteres que no sea una palabra reservada, aunque suelen utilizarse letras mayúsculas.
Veamos otro ejemplo, en este caso para definir una interfaz genérica:

```
public interface Pareja<K, V> {
    K getKey();
    V getValue();
}
```

En el momento de la declaración y/o instanciación de un objeto de una que utilice un tipo genérico indicaremos el tipo de datos que finalmente utilizará. Por ejemplo, en el caso anterior de la clase Pila y suponiendo que quisiéramos crear una Pila donde sus elementos sean de tipos Strings se haría de la siguiente forma:

```
Pila<String> pila = new Pila<String>();
```

De este modo, todos los atributos y métodos definidos con el tipo genérico se adaptarán al tipo String.

A veces puede ser que queramos limitar los tipos de datos que pueden ser empleados cuando hacemos uso de Generics. Java proporciona un mecanismo para llimitar los *`Generics`* mediante los *bounded type*, que no es más que añadir extends y un tipo de datos base mínimo. Por ejemplo, si queremos que nuestra Pila solo trabaje con números (pero cualquier tipo de números, enteros, reales, etc...) haríamos:

```
public class Pila<T extends Number> {
...
}
```

Limitaciones

Los *`Generics`* tienen algunas restricciones que hay que tener en cuenta:

- No se pueden instanciar tipos genéricos con tipos de datos primitivos.
- No se pueden crear instancias de los parámetros del tipo genérico.
- No se pueden declarar campos *`static`* del tipo genérico.
- No se pueden utilizar *`casting`* o instanceof con tipos parametrizados.
- No se pueden crear, capturar o lanzar tipos parametrizados que extiendan a Throwable .