Profundizando en las clases y métodos genéricos.

Dos o más parámetros de tipo. (I)

```
public class Util {
public static <T,M> int sumaDeLongitudes (T[] a, M[] b)
{
	return a.length+b.length;
}
}
```

Si un método genérico necesita tener dos o más parámetros genéricos, podemos indicarlo separándolos por comas. En el ejemplo anterior se suman las longitudes de dos arrays que no tienen que ser del mismo tipo.

Dos o más parámetros de tipo. (II)

```
class Terna <A,B,C>  {
    A a; B b; C c;
    public Terna (A a, B b, C c)
      {this.a=a; this.b=b; this.c=c;}
    public A getA () { return a; }
    public B getB () { return b; }
    public C getC () { return c; }
}
```

Si una clase genérica necesita tener dos o más parámetros genéricos, podemos indicarlo separándolos por comas. En el ejemplo anterior se muestra una clase que almacena una terna de elementos de diferente tipo base que están relacionados entre sí.

Dos o más parámetros de tipo. (II)

```
Integer[] a1={0,1,2,3,4};
Double[] a2={0d,1d,2d,3d,4d};
Util.<Integer,Double>sumaDeLongitudes(a1,a2);
```

Usar un método o una clase con dos o más parámetros genéricos es sencillo, a la hora de invocar el método o crear la clase, se indican los tipos base separados por coma.

Métodos con tipos adicionales.

```
class Util <A>  {
    A a;
    Util (A a) { this.a=a; }
    public <B> void Salida (B b) {
          System.out.println(a.toString() + b.toString());
}
```

Una clase genérica puede tener unos parámetros genéricos, pero si en uno de sus métodos necesitamos otros parámetros genéricos distintos, no hay problema, podemos combinarlos.

Inferencia de tipos. (I)

```
Integer[] a1={0,1,2,3,4};

Double[] a2={0d,1d,2d,3d,4d};

Util.<Integer,Double>sumaDeLongitudes(a1,a2);

Util.sumaDeLongitudes(a1,a2);
```

No siempre es necesario indicar los tipos a la hora de instanciar un método genérico. A partir de Java 7, es capaz de determinar los tipos a partir de los parámetros. Las dos expresiones de arriba serían válidas y funcionarían. Si no es capaz de inferirlos, nos dará un error a la hora de compilar.

Inferencia de tipos. (II)

```
Integer a1=0; Double d1=1.3d; Float f1=1.4f;
Terna <Integer,Double,Float> t= new Terna<>(a1,d1,f1);
```

A partir de Java 7 es posible usar el operador diamante (“<>”) para simplificar la instanciación o creación de nuevos objetos a partir de clases genéricas.

¡¡Cuidado, esto es solo a partir de Java 7!!

Limitación de tipos.

```
public class Util {
public static <T extends Number> Double sumar (T t1, T
t2){
 return new Double(t1.doubleValue() + t2.doubleValue());
}
}
```

Se pueden limitar el conjunto de tipos que se pueden usar con una clase o método genérico usando el operador “extends”. El operador extends permite indicar que la clase que se pasa como parámetro genérico tiene que derivar de una clase específica. En el ejemplo, no se admitirá ninguna clase que no derive de Number, pudiendo así realizar operaciones matemáticas.

Paso de clases genéricas por parámetro.

```
public class Ejemplo <A> {
public A a;
}
…
void test (Ejemplo<Integer> e) {...}
```

Cuando un método tiene como parámetro una clase genérica (como es el caso del método test del ejemplo), se puede especificar cuál debe ser el tipo base usado en la instancia de la clase genérica que se le pasa como argumento. Esto permite, entre otras cosas, crear diferentes versiones de un mismo método (sobrecarga): dependiendo del tipo base usado en la instancia de la clase genérica se ejecutará una versión u otra.

Paso de clases genéricas por parámetro. Wildcards. (I)

```
public class Ejemplo <A> {
public A a;
}
…
void test (Ejemplo<?> e) {...}
```

Cuando un método admite como parámetro una clase genérica, en la que no importa el tipo de objeto sobre el que se ha creado, podemos usar el interrogante para indicar “cualquier tipo”.

Paso de clases genéricas por parámetro. Wildcards. (II)

```
public class Ejemplo <A> {
public A a;
}
…
void test (Ejemplo<? extends Number> e) {...}
```

También es posible limitar el conjunto de tipos que una clase genérica puede usar, a través del operador extends. El ejemplo anterior es como decir “cualquier tipo que derive de Number”.