# Conocimiento : Generics

**Definición**

En el momento de escribir una clase se debe conocer con que tipo de datos va a interactuar, sin embargo esto no siempre es conocido. Por lo tanto se debería poder definir una clase con la ayuda de un “contenedor” al cual nos referimos como si fuera el tipo sobre el que opera la clase.

La definición actual de la clase es creada una vez que declaramos un objeto en particular. Los tipos genéricos, también llamados tipos parametrizados, permiten definir un tipo sin especificar todos los tipos que éste usa. Estos serán suministrados como parámetros en punto de instanciación. Los cambios son realizados por tanto en tiempo de compilación.

Los tipos parametrizados se utilizan especialmente para implementar tipos abstractos de datos: pilas, colas, listas, conjuntos y otros que permiten almacenar distintos tipos de elementos según sean instanciados en tiempo de compilación.

El principal uso de los Generics es el de tipar a las collections, antes de la versión 1.5 solamente los arrays eran tipados, se podía declarar algo del tipo:

Persona [] agenda = new Persona [10];

Donde el compilador solamente aceptará Persona(o alguna subclase de Persona) como elementos de agenda, si intentamos agregar cualquier otro tipo de objeto obtendremos una excepción.

Pero este mismo tipado no existía para las collections, en ellas no podíamos definir que las estructuras fueran de un solo tipo, cualquier objeto de tipo collection podía albergar cualquier objeto que extendiese de Object (es decir CUALQUIER clase menos las primitivas), con lo cual para, por ejemplo obtener la primera entrada de un objeto cualquiera tendríamos que hacer un casting manual cada vez que obteníamos un elemento, porque no podíamos asegurar que los elementos fueran del tipo indicado; y considerando, además, el tratamiento de errores que habría que hacer por el casting.

Algunos de los beneficios de los tipos genéricos en Java son:

* Comprobación estricta de tipos manteniendo la misma flexibilidad que el enlazado dinámico. Con tipos genéricos se puede alcanzar un polimorfismo similar al usado en el ejemplo anterior pero con una comprobación estricta de tipos que permite detectar errores en tiempo de compilación. El compilador conoce que los tipos de listas son diferentes porque contienen distintos elementos. Los errores al ser detectados en tiempo de compilación son mucho más fáciles de detectar que los errores en tiempo de ejecución.
* No es necesaria la comprobación de tipos en tiempo de ejecución, lo que redunda en un código con menos castings y por lo tanto más legible y menos propenso a errores. En lugar de confiar en la memoria del usuario los parámetros marcan el tipo de los elementos obtenidos de la lista.
* Los tipos genéricos garantizan que las listas contienen solo un conjunto homogéneo de elementos eliminando los errores derivados de la aparición de listas heterogéneas.
* Hace que el código sea menos ambiguo y más fácil de mantener.

**Sintaxis**

Para poder explicitar un tipo de dato parametrizado se explicita de la siguiente manera:

NombreClase<Tipo de dato>

Para comprender la mejora de los generics, veremos un ejemplo de colección que permite agregar animales, primero sin usar generics y después usando generics:

* Antes:

List granja = new ArrayList ();

granja.add (perro);

Animal a = (Animal)granja.get(0);

granja.add(“hola”);

Animal a = (Animal)granja.get(1); // ERROR !!!

* Después:

**List<Animal> granja = newArrayList<Animal> ();**

granja.add(perro);

Animal a = granja.get(0); // SIN CASTING

granja.add(“hola”); // ERROR !!!

**Clase Genérica**

En una clase genérica, el tipo de parámetro (T) aparece en la cabecera al declarar la clase, pero no en el constructor. Para entender este proceso veamos el siguiente ejemplo:



Para invocar el uso de esta clase basta con definir en la llamada a qué tipo de dato nos referiremos, que puede ser una clase o interfaz, pero no valor primitivo, por ejemplo:



En este ejemplo primero se invoca la clase genérica con parámetros de tipo Integer y después se vuelve a invocar, pero ahora con Carácter.

Veamos el resultado al ejecutar este programa:



**Constructores genéricos**

En las clases genéricas los constructores mantienen su estructura inalterable, es decir no es necesario agregar ningún elemento a la sintaxis, ya que la definición de genericidad lo da la cáscara de la clase.

Veamos el ejemplo anterior:

 public ClaseGenerica(T nombre)

 {

 this.nombre= nombre;

 }

En este constructor el tipo de dato T indica que puede recibir cualquier tipo complejo.

**Métodos genéricos**

Es posible definir variables de tipo específicas para un único método dentro de la clase, por ejemplo:



Esta variable de tipo puede utilizarse solo en este método.