

# UTX Varios

## Clases y tipos genéricos en Java

En **[Java](http://jonsegador.com/tag/java/)**, cuando definimos una nueva clase, **debemos conocer el tipo de dato** con el que trabajaremos. Si queremos realizar una operación específica dentro de esta nueva clase, **sea cual sea el tipo de datos** que va a recibir, podemos hacer uso de los **tipos genéricos**. Este tipo genérico asumirá el tipo de dato que realmente le pasaremos a la clase.

Mejor con un ejemplo:

|  |
| --- |
| **class** ClaseGenerica<T> {  T obj;    **public** ClaseGenerica(T o) {  obj = o;  }    **public** **void** classType() {  System.out.println("El tipo de T es " + obj.getClass().getName());  }  }    **public** **class** MainClass {  **public** **static** **void** main(String args[]) {  *// Creamos una instancia de ClaseGenerica para Integer.*  ClaseGenerica<Integer> intObj = **new** ClaseGenerica<Integer>(88);  intObj.classType();    *// Creamos una instancia de ClaseGenerica para String.*  ClaseGenerica<String> strObj = **new** ClaseGenerica<String>("Test");  strObj.classType();  }  } |

Notas:

* T es el tipo genérico que será reemplazado por un tipo real.
* T es el nombre que damos al parámetro genérico.
* Este nombre se sustituirá por el tipo real que se le pasará a la clase.

El resultado será el siguiente:

|  |  |
| --- | --- |
| 1  2 | El tipo de T es java.lang.Integer  El tipo de T es java.lang.String |

Hay que tener en cuenta que **los generics de java solo funcionan con objetos**. El código siguiente nos mostrará un error:

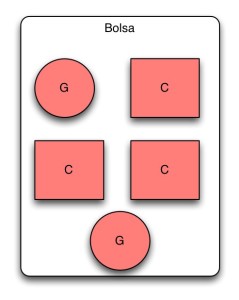
ClaseGenerica<int> myOb = **new** ClaseGenerica<int>(53);   
*// Error, can't use primitive type*

Existen una serie de **convenciones para nombrar a los genéricos**:

* E – Element (usado bastante por Java Collections Framework)
* K – Key (Llave, usado en mapas)
* N – Number (para números)
* T – Type (Representa un tipo, es decir, una clase)
* V – Value (representa el valor, también se usa en mapas)
* S,U,V etc. – usado para representar otros tipos.

### 1.1 Uso de Java Generics

Veremos cómo se usan los Java Generics o llamadas simplemente clases Genéricas. Construiremos  la clase Bolsa que es una clase sencilla que nos permitirá almacenar objetos de varios tipos.

[](http://www.arquitecturajava.com/wp-content/uploads/bolsa.jpg)

Esta clase tendrá un límite de objetos a almacenar. Alcanzado el límite no se podrán añadir más. Vamos a ver su código fuente:

import java.util.ArrayList;

import java.util.Iterator;

public class Bolsa implements Iterable{

private ArrayList lista;

private int tope;

public Bolsa(int tope) {

//super();

lista= new ArrayList();

this.tope = tope;

}

public void add(Object objeto ) {

if (lista.size()<=tope) {

lista.add(objeto);

}else {

throw new RuntimeException("no caben mas");

}

}

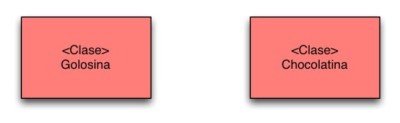
public Iterator iterator() {

return lista.iterator();

}

}

En nuestro caso vamos a disponer de dos clases con las cuales rellenar la bolsa. La clase Golosina y la clase Chocolatina.

[](http://www.arquitecturajava.com/wp-content/uploads/chocolatina-golosina.jpg)

Vamos a ver su código fuente:

public class Golosina {

private String nombre;

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public Golosina(String nombre) {

super();

this.nombre = nombre;

}

}

public class Chocolatina {

private String marca;

public String getMarca() {

return marca;

}

public void setMarca(String marca) {

this.marca = marca;

}

public Chocolatina(String marca) {

super();

this.marca = marca;

}

}

Creamos un sencillo programa que llene la Bolsa de Chocolatinas y Golosinas para luego recorrer los elementos que están en la bolsa y sacarlos por pantalla.

public class Principal {

public static void main(String[] args) {

Bolsa bolsa= new Bolsa(5);

Chocolatina c= new Chocolatina("milka");

Chocolatina c1= new Chocolatina("milka");

Chocolatina c2= new Chocolatina("ferrero");

Golosina g1= new Golosina("gominola");

Golosina g2= new Golosina("chicle");

bolsa.add(c);

bolsa.add(c1);

bolsa.add(c2);

bolsa.add(g1);

bolsa.add(g2);

for (Object o: bolsa) {

if( o instanceof Chocolatina) {

Chocolatina chocolatina= (Chocolatina)o;

System.out.println(chocolatina.getMarca());

}else {

Golosina golosina= (Golosina)o;

System.out.println(golosina.getNombre());

}

}

}

}

El programa funcionará correctamente, pero nos podremos dar cuenta que **resulta bastante poco amigable la estructura if /else en la cual se chequean cada uno de los tipos a la hora de presentarlo por pantalla.**

### 1.2 Java Generics

Para solventar este problema podemos construir una clase Genérica. Este tipo de clase nos permitirá definir una Bolsa de un tipo concreto. Puede ser una bolsa de Golosinas o una bolsa de Chocolatinas **pero NO de las dos cosas  a la vez**. Esto en un principio puede parecer poco flexible pero si nos ponemos a pensar cuando programamos solemos imprimir una lista de Facturas o una lista de Compras no una lista mixta. Así pues el enfoque parece razonable. Vamos a ver el código fuente y comentarlo:

import java.util.ArrayList;

import java.util.Iterator;

public class Bolsa<T> implements Iterable<T>{

private ArrayList<T> lista= new ArrayList<T>();

private int tope;

public Bolsa(int tope) {

super();

this.tope = tope;

}

public void add(T objeto ) {

if (lista.size()<=tope) {

lista.add(objeto);

}else {

throw new RuntimeException("no caben mas");

}

}

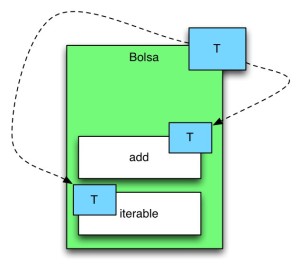
public Iterator<T> iterator() {

return lista.iterator();

}

}

La clase es un poco peculiar ya que al no saber de entrada de qué tipo va a ser la bolsa **debemos declarar un tipo Genérico T a nivel de clase y que será repetido en cada uno de los métodos que lo usen.**

[](http://www.arquitecturajava.com/wp-content/uploads/tipoT-e1401809865872.jpg)

De esta manera cuando construyamos un objeto de esta clase será el momento de especificar el tipo de Bolsa que deseamos. En el siguiente ejemplo hemos elegido “Chocolatina” como tipo para la Bolsa. De esta manera la bolsa solo admitirá este tipo de objetos.

public class Principal {

public static void main(String[] args) {

Bolsa<Chocolatina> bolsa= new Bolsa<Chocolatina>();

Chocolatina c= new Chocolatina("milka");

Chocolatina c1= new Chocolatina("milka");

Chocolatina c2= new Chocolatina("ferrero");

bolsa.add(c);

bolsa.add(c1);

bolsa.add(c2);

for (Chocolatina chocolatina : bolsa) {

System.out.println(chocolatina.getMarca());

}

}

}

<http://www.arquitecturajava.com/uso-de-java-generics/>

## Copiar Arrays en Java

### 2.1 Copiar arrays en Java

Muchas veces nos vamos a encontrar ante la disyuntiva de tener que copiar los elementos de un [array](http://www.w3api.com/wiki/Java:Array" \t "Array Java) en otros. Normalmente será para manipular el contenido del mismo guardando en uno de los arrays los datos originales.

Asumida ya la situación, lo primero que se nos ocurriría, independientemente del lenguaje en el que nos encontremos, será el montar un algoritmo que recorriendo el primer [array](http://www.w3api.com/wiki/Java:Array" \t "Array Java) vaya copiando dichos elementos en el segundo.

En [Java](http://www.manualweb.net/tutorial-java/" \t "Manual de Java) nos quedaría un código como este:

for (int x=0;x<aOrigen.length;x++)

aDestino[x] = aOrigen[x];

En este sentido nada que objetar, ya que es muy buena práctica de programación. Pero todo buen programador tiene que tener en mente el concepto de "reutilización". No "reutilización" cómo copia del código, sino "reutilización" pensando en que alguien ya puede haberse encontrado el problema y haberle dado ya una solución.

Para poder reutilizar tenemos que ser conscientes de lo que el entorno en el que estamos nos ofrece. Y en el caso de [Java](http://www.manualweb.net/tutorial-java/" \t "Manual de Java), es la librería del sistema la que nos ofrece una función para la copia de arrays. Como vemos en el siguiente código:

[System](http://www.google.com/search?hl=en&q=allinurl%3ASystem+java.sun.com&btnI=I'm Feeling Lucky).arrayCopy(aOrigen,inicioArrayOrigen,aDestino,

inicioArrayDestino,numeroElementosACopiar);

Hay que tener cuidado [la función arrayCopy](http://www.w3api.com/wiki/Java:System.arraycopy()" \t "Método Java arraycopy) ya que esta nos puede devolver las siguientes excepciones: [IndexOutOfBoundsException](http://www.w3api.com/wiki/Java:IndexOutOfBoundsException" \t "Excepción Java IndexOutOfBoundsException) si intentamos copiar fuera del área reservado para el array, [ArrayStoreException](http://www.w3api.com/wiki/Java:ArrayStoreException" \t "Excepción Java ArrayStoreException) si intentamos copiar arrays de diferente tipo o [NullPointerException](http://www.w3api.com/wiki/Java:NullPointerException" \t "Excepción Java NullPointerException) si alguno de los array es nulo (es decir, no inicializado).

### **2.2 Copiar dos arrays en uno con Java**

A partir de dos arrays, copiar el contenido de ambos dentro de un tercer array.

Lo primero será definir los dos arrays de origen:

int a1[] = {1,2,3,4,5};

int a2[] = {6,7,8,9,10};

Y posteriormente el array destino. Hay que tener en cuenta que el tamaño del array destino tiene que ser lo suficientemente grande como para albergar el contenido de los dos arrays origen.

Para ello le damos como tamaño la suma de los dos arrays de origen:

int a[] = **new** int[a1.length+a2.length];

En la copia de arrays nos apoyaremos en el método [Java](http://www.manualweb.net/tutorial-java/" \t "Manual de Java)  [arrayCopy](http://www.w3api.com/wiki/Java:System.arraycopy()" \t "Método Java arraycopy).

Los parámetros que recibe el método arrayCopy son:

* Array origen.
* Posición inicial del array origen.
* Array destino.
* Posición inicial en el array de destino.
* Número de elementos a copiar del array origen al array destino.

El siguiente paso será copiar el primer array en el array destino:

System.arraycopy(a1, 0, a, 0, a1.length);

Vemos que del array origen copiamos desde la posición 0, al igual que el array destino. El número de elementos a copiar es igual al tamaño del array de origen.

El tercer paso será copiar array en el array destino. Hay que tener cuidado, ya que en el array destino ya tenemos cargado el primer array.

System.arraycopy(a2, 0, a, a1.length, a2.length);

Lo que vemos es que del array de origen se copia desde el primer elemento, el cero. En el caso del array destino nos tendremos que posicionar en el elemento siguiente al último elemento del primer array. Esto nos lo da el tamaño del primer array. Es por ello que utilizamos a1.length. El número de elementos sigue siendo los elementos del segundo array.

## 3. Listas en Java

Una **lista** es una secuencia de elementos dispuestos en un cierto orden, en la que cada elemento tiene como mucho un predecesor y un sucesor. El número de elementos de la lista no suele estar fijado, ni suele estar limitado por anticipado. Representaremos la estructura de datos de forma gráfica con cajas y flechas. Las cajas son los elementos y las flechas simbolizan el orden de los elementos.

Una lista

La estructura de datos deberá permitirnos determinar cuál es el primer elemento y el último de la estructura, cuál es su predecesor y su sucesor (si existen de cualquier elemento dado). Cada uno de los elementos de información suele denominarse nodo.

La lista también puede representarse de forma simbólica escribiendo sus elementos separados por comas y encerrados entre corchetes. Por ejemplo:

["rojo","verde","azul","amarillo"]

Las listas admiten ciertas operaciones como son insertar un nodo adicional, borrar un nodo, etc. En función de la forma de insertar nuevos elementos y acceder a los existentes tendremos distintos tipos de listas. Veamos ahora qué operaciones básicas se pueden realizar sobre las listas.

Hay ciertas operaciones básicas sobre una lista como:

* **EsVacia** Averiguar si la lista está vacía.
* **Insertar** Añade un elemento al principio de la lista.
* **Primero** Obtener el valor del primer elemento de la lista, también llamado cabeza.
* **Resto** Devuelve el trozo de lista resultado de eliminar el primer elemento de la lista.
* **Borrar** Borrar el primer elemento de la lista.
* **etc.**

## 4. Pilas

Una **pila** (stack en inglés) es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»).

