

# Monografico Java 5.0

Nuevas características en el lenguaje Java

Pedro Del Gallego Vida

[pedro.delgallego@javahispano.org](mailto:pedro.delgallego@javahispano.org)

# Novedades

- La sentencia For/in
- Generics
- Tipos Enumerados
- Autoboxing - unboxing
- Varargs
- Anotations
- Formatting

# La sentencia For / in

- La sentencia For/in es un nuevo tipo de bucle, que nos permite iterar sobre diferentes tipos de datos.
- Iteraciones sobre arrays
- Iteraciones sobre Collections
- Como hacer que nuestras clases funcionen con el bucle for in

# La sentencia for/in (II)

- Estructura
  - for (declaration : expression)  
statement
- Que esta pasando realmente
  - for ( Iterator<E> #i = (expression).iterator( );  
#i.hasNext( ); ) {
  - declaration = #i.next( );
  - statement
  - }

# For in sobre arrays

- Podemos iterar perfectamente sobre arrays de tipos primitivos como de objetos.
- Por ejemplo
  - `Int numeros[ ] = new int[4]`
  - `For (int numero : numeros) ....`

# For in sobre Collections

- Podemos iterar sin problemas sobre los distintos tipos de Collections.
- Según el tipo de Collection que usemos puede variar el orden y si existen repeticiones o no (por ejemplo listar los elementos de un LinkedList podría ser diferente de Set)
- No serán necesarios los casting dentro del bucle.

# For in y nuestras clases

- Podemos hacer que nuestras clases funcionen con el nuevo bucle si implementa tanto la interfaz `Iterator<E>` como la interfaz `Iterable<E>`
- Podemos hacer clases que sean iterables con el bucle `for in` si extienden a alguna clase `Collection`.

# Genericos

- Que son los generics.
- Uso de generics al definir listas.
- Uso de generics al definir Hashmap
- Creando nuestros propios generics

# Que son los generics

- Un generico es una forma de marcar que un determinado objeto sera de una clase determinada o que vendra definido por una plantilla
- Es una forma de hacer mas seguros nuestros programas al hacer un tipado mas fuerte en tiempo de compilacion.
- Permite eliminar algunos castings

# Uso de generics al definir una lista (o una clase Collection)

- Podemos definir una variable de tipo Collection de la siguiente forma
  - `LinkedList<E> lista = new LinkedList<E>();`
  - Donde E representa cualquier tipo de primitiva u objeto.
- El compilador se encargara de realizar las comprobaciones de que estamos insertando el tipo adecuado.

# Uso de generics al definir un hashmap

- Podemos usar los generics para tipar los objetos hashmap
  - `Map<Integer, Integer> squares = new HashMap<Integer, Integer>();`
  - `Map<String, Object> map = new HashMap<String, Object>( );`
- Nos permite crear un fuerte tipado durante el tiempo de ejecución.

# La sentencia for/in junto a generics

- Podemos usar estas dos características de forma conjunta para crear bloques robustos y mas legibles ya que usaran el bucle for in , no tendran que hacer casting y podra indicarse dentro de la sentencia for el tipo que se quiere obtener
  - `LinkedList<Usuario> lista=new LinkedList<Usuario>()`
  - `For (Usuario user : lista) {.....}`

# Parametrizando la entrada y salida

- Se puede parametriza la entrada de un método
  - `private void printListOfStrings(List<String> list )`
- Al igual podemos parametrizar la salida de un método
  - `private List<String> getListOfStrings(`

# Plantillas

- Podemos parametrizar según una plantilla por ejemplo :
  - `public void drawAll(List<? extends Shape> shapes) {...}`
- Podemos parametrizar listas parametrizadas :
  - `list<list<? Extends shape>>`

# Limitaciones de las plantillas

- Existen ciertas limitaciones al usar plantillas
  - ```
public void addRectangle(LinkedList <? extends Shape> shapes) {  
    shapes.add(new Rectangle()); // compile-time error  
}
```
  - Genera un error ya que no puede asegurarse de que `Rectangle` sea una subclase de `Shape` entonces no es seguro llevar la operación.
- Otras

# Escribiendo nuestros generics

- Podemos escribir nuestros propios tipos de generics, usando un tipo base o una plantilla
- Son utiles para collecciones ordenadas, clases contenedoras.
- No se permite la parametrización de miembros estaticos
- Podemos extender o parametrizar mas nuestros genéricos

# Tipos enumerados

- Que son los tipos enumerados
- Crear un tipo enumerados
- Iterar sobre un tipo enumerado
- Añadir metodos un tipo enumerado
- Extender un tipo enumerado
- Implementar interfaces con un tipo enumerado

# Que son los tipos enumerados

- Los tipos enumerados vienen a cubrir el hueco un faltaba en java.
- Vienen para sustituir al patron (Effective java item 16)
- Los tipos enumerados son clases, y extienden la interfaz `java.lang.Enum`
- Los valores de los tipos enumerados no son entereros.
- Los tipos enumerados son publicos, staticos y “finales”

# ¿Que son los tipos enumerados? (II)

- Pueden ser comparados con `==` y con `equals()`, el orden bien determinado en la posición en que se definen
- Los tipos enumerados sobrescriben el método `toString()`. Devuelven una cadena con su nombre.
- Definen un método llamado `values` que nos devuelve todos los valores.
- Un clase enumerada puede ser usada de la misma forma que las demás.

# Crear un tipo enumerado

- Nueva palabra del lenguaje : “enum”
- Los tipos enumeados no tienen constructores publicos
- Podemos crear una clase enum “inline”, pero el compilador la tomara como static

# Iterar sobre tipos enumerados

- Podemos iterar sobre un tipo enumerado usando el metodo `values` que nos devuelve la colección de valores posibles ordenados de menor a mayor y usando un array podemos iterar sobre estos.

# Añadir métodos a un tipo enumerado

- Los tipos enumerados al ser clases pueden contener métodos.
- Si algun metodo dentro del tipo enumerado es un constructor este debera ser privado.

# Implementar interfaces con un tipo enumerado

- Como los tipos numerados pueden contener metodos entonces pueden implementar una interfaz.

# Autoboxing unboxing

- Convertir tipos primitivos en tipos wrapper
- Convertir tipos wrapper en tipos primitivos
- Incrementar decrementar tipos wrapper

# Convertir tipos primitivos en tipos wrapper

- Tiger convierte los tipos primitivos a su tipo wrapper correspondiente bajo demanda, esto quiere decir que no hay por que complicarse con casting de unos a otros
-

# Convertir tipos wrapper en tipos primitivos

- Tiger convierte un tipo wrapper a su primitivos correspondiente bajo demanda, esto quiere decir que no hay por que complicarse con casting de unos a otros

# Incrementar decrementar tipos warpper

- Con el nuevo sistema de autoboxing/unboxing podemos incrementar o decrementar con los operadores ++ y -- a esto se le llama autounboxing.

# Varargs

- Crear una lista variable de argumentos
- Iterar sobre una lista variable de argumentos
- Listas de 0 a N argumentos

# Crear un lista variable de argumentos

- Podemos crear funciones que admitan de 0 a N argumentos.
- Para ello usaremos los puntos suspensivos. “ Tipo .... nombre de la variable ”.
- Solo puede existir un parametro de “...”

# Iterar sobre una lista de argumentos

- Ya que la lista de argumentos viene dada por `String ... args` que es una colección sobre la que podemos desplazarnos.

# Listas de 0 a N argumentos

- En lista de argumentos variables pueden existir de 0 a N argumentos
- Si hay cero argumentos el valor sera null

# Annotations

- Que son las annotations
- Anotaciones estandard
- Crear custom annotations
- Annotating Annotations
- Defining an Annotation Type's Target
- Setting the Retention of an Annotation Type
- Documenting Annotation Types
- Setting Up Inheritance in Annotations

# Que son las anotaciones

- Las anotaciones sirven para crear metadatos dentro de las clases.
- Las anotaciones son una forma de especificar cosa acerca de
  - Clases
  - Metodos
  - Atributos
  - Interfaces
  - Anotaciones

# Que son las anotaciones (II)

- Para indicar que vamos a realizar una anotacion usaremos el signo reservado @
- Podemos distinguir tres tipos de anotaciones
  - Documentalistas
  - Tiempo de compilación
  - Tiempo de ejecución

# Conjunto de anotaciones en JDK 5.0

- Existe un conjunto predefinido de anotaciones:
  - `@Override`
  - `@Deprecated`
  - `@SuppressWarnings`
  - `@Documented` -- Meta-anotación
  - `@Target` -- Meta-anotación
  - `@Retention` -- Meta-anotación

# @override

- Indica que el metodo al que se lo ponemos sobreescribira a otro metodo, esn caso de que no fuese asi entonces el compilador nos avisaria.

# @deprecated

- Esta anotación marca un método como deprecated, y en caso de que lo usemos nos mostrará un warning en tiempo de compilación

# @supresswarnings

- Oculta durante la compilacion los warnings al tipo

# Crear nuestras propias anotaciones

- Podemos crear nuestras propias anotaciones mediante el uso de `@interface`
- La definicion sera del tipo par-valor. Por ejemplo:
  - `String value();`
  - `String menssagge();`
  - `Int code();`

# Crear nuestras propias anotaciones (II)

- Podemos marcar un valor por defecto para cada par-valor mediante el uso de la palabra reservada `default`
- Dentro de una `@interface` podemos definir una clase
- La `@interface` puede descubrir extiende la interfaz `java.lang.annotation.Annotation` por lo cual no puede extender nada mas.

# Anotaciones sobre anotaciones

- Una anotación sobre una anotación se llama meta-anotación
- Existen cuatro tipos estandar de meta-anotaciones
  - Target : marcan que puede ser anotado con esta anotación
  - Retention : indica que tipo de anotación es.
  - Documentation :
  - Inherited : marcan si se heredan las anotaciones

# @target

- Es una meta- anotación, que nos indica a que se puede aplicar la anotacion a la que se refiere.
- Puede indicar :
  - TYPE, // Clases, interfaces o enumerados
  - FIELD, // Campos, incluidos los enumerados
  - METHOD, // Method (does not include constructors)
  - PARAMETER, // parametro de un metodo
  - CONSTRUCTOR, // Constructor
  - LOCAL\_VARIABLE, // Local variable or catch clause
  - ANNOTATION\_TYPE, // meta- anotaciones
  - PACKAGE // Sobre el paquete.

# @Retention

- Es una meta- anotación, que indica hasta que punto debe guardarse la anotación.
- Puede tomar los siguientes valores. :
  - SOURCE, // son descartadas por el compilador
  - CLASS, // se guardan en el archivo class pero son ignoradas por la VM
  - RUNTIME // se guardan en el archivo class y son leídas por la VM

# Formatos

- Que es un formato
- Como crear un formato

# Que es un Formato

- Los formatos sirven para crear plantillas y simplificar el formateo de una salida
- Se asemejan a los formatos de C
- Se usa el simbolo % para indicar que se va introducir un valor.

# Como crear un formato

- La clase `Formatter` nos permite crear plantillas para la salida de nuestros datos.
- Tambien podemos usarlo `System.out.Format`
- Para dar el formato usaremos el metodo `format` :
  - `public Formatter format(String format, Object... args);`
  - `public Formatter format(Locale l, String format, Object... args);`

# Escribir salidas formateadas

- Para escribir una salida formateada usaremos el metodo format
- Podemos indicar que el lugar que debe ocupar una variable poniendo el smbolo %