Clase 6-anotaciones

miércoles, 8 de octubre de 2025 14:57

Son una especie de meta-dato.
Dan información al código sin cambiar la semántica.
Son verificables por el compilador.
Es info extra testeada y verificada en compilación.
Dan info descriptiva que no se puede hacer en java.
Da info para las herramientas, el código nop se afecta directamente por las anotaciones.
El programador dice que hay que hacer y la herramienta lo hace automaticamente, produce el codigo.
Ej: override
Notación del compilador
Es cpomo que se usan para que nos de errores o warnings.
Son ayudas para el desarrollador.

Como declarar anotaciones:
Es como declarar una interfaz pero se le pone un @ antes de interface.
Se compilan a archivos.class igual que cualquier cosa.

```
public @interface SolicitudDeMejora{
    int id();
    String resumen();
    String ingeniera() default "[no asignado]";
    String fecha() default "[no implementado]";
}
```

Definición de la ANOTACIÓN SolicitudDeMejora
El cuerpo de la declaración de las anotaciones contiene **declaraciones de elementos** o **parámetros** que permiten especificar valores para las anotaciones.
Los programas o herramientas usarán los valores de los elementos o parámetros para procesar la anotación.

Fecha e ingeniera tienen valores por default y no necesitan valores.

```
@ SolicitudDeMejora(
id = 2868724,
resumen= "Habilitado para viajar en el tiempo",
ingeniera = "Elisa Bachofen",
fecha = "6/10/2024" )
public static void viajarEnElTiempo(Date destino) { }
```

Es como un comentario más complejo.
El compilador verifica que siga la estructura, que esté bien usada. Pero no hace nada como tal la anotación.

Para las anotaciones con un **único elemento** se puede usar el nombre **value** y de esta manera cuando anotamos un elemento omitimos el nombre del elemento seguido del signo igual (=).

```
/*Asocia un copyright al elemento anotado*/
public @interface Copyright {
    String value();
}
```

Definición de la ANOTACIÓN Copyright

El elemento de nombre **value** es el único que no requiere el uso de la sintaxis elemento-valor para anotar declaraciones, alcanza con especificar el valor entre paréntesis:

```
@Copyright("2022 Sistema de Auditoría de Juegos de Azar")
public class MaquinasDeJuego{ }
```

La clase MaquinasDeJuego está anotada con la anotación Copyright

Anotaciones *Marker*

Las **Anotaciones *Markers* NO contienen elementos**, son útiles para marcar elementos de una aplicación con algún propósito.

Definición de las anotaciones *markers* "InProgress" y "Test":

```
/*Indica que el elemento anotado está sujeto a cambios, es una versión preliminar */
public @interface InProgress { }
```

La anotación **@Test** la usaremos para realizar **testeos** que se ejecutan **automáticamente**. Cuando un método falla se disparan excepciones.

```
import java.lang.annotation.*;
/**
* Indica que el método anotado es un método de testeo.
* Se usa sólo en métodos estáticos y sin argumentos.
*/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
}
```

¿Puede el compilador garantizarlo?

El target es quie solo se puede aplciar a metodos
El retention policy
No puede garantizarlo el compilador. No tiene ninguna data para chequear quje sea así

La Anotación @ExceptionTest

```
package anotaciones;
import java.lang.annotation.*;
/**
* Indica que el método anotado es un método de testeo
* que se espera dispare la excepción consignada como parámetro
*/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Exception> value();
}
```

El tipo del elemento **value() es un objeto Class** para cualquier **subclase de Exception**

Las **pruebas que tienen éxito** sean las que arrojan **una excepción particular**, por eso esta anotación tiene parámetros.

```
package anotaciones;
public class Ejemplo2 {
@ExceptionTest(ArithmeticException.class)
public static void m1() {
  int i = 0;
  i = i / i;
}
@ExceptionTest(ArithmeticException.class)
public static void m2() {
  int[] a = new int[0];
  int i = a[1];
}
@ExceptionTest(ArithmeticException.class)
public static void m3() { }
}
```

La primero está bien pq da aritmeticca, la segunda no pq es out opf bpounds.

```
Class<? extends Exception> excType = m.getAnnotation(ExceptionTest.class).value();
if (excType.isInstance(exc)) {
   passed++;
} else {
```

¿La excepción disparada es la esperada?

Puedo pedir los parámetros de la anotación. En este caso va a darme el tipo de excepción que se disparó.

Consideremos que **las pruebas que pasan** son las que **disparan** al **menos una excepción de un conjunto de excepciones.**

¿Cómo haríamos?

```
package anotaciones;
import java.lang.annotation.*;
/**
* Indica que el método anotado es un método de testeo
* que se espera dispare la excepción consignada como
* parámetro
*/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
Class<? extends Exception> [] value();
}
```

El tipo del elemento **value() es un arreglo de objetos Class** para cualquier **subclase de Exception**

```
package anotaciones;
public class Ejemplo3 {
@ExceptionTest({ IndexOutOfBoundsException.class, NullPointerException.class})
public static void doublyBad() {
        List<String> list = new ArrayList<>();
        list.add(5, null);
}
```

No estaba escuchando xd pero esto es para cuando dispara alguna de las dos excepcioens

EXCEPCIONESÑ
Errores qiue pasan durante la ejecución de un programa

checked Exception o Verificables en Compilación:Los que son verificables en compilación son los que se pueden recuperar
Los runtime son internos, malos malos, no verificables en compilación. Error de codigo
Error: externos, problemas de memoria, etc