Lección: Genéricos.

En cualquier proyecto de software, los errores son algo normal. Sin embargo la adecuada planificación, programación y las pruebas pueden ayudar a reducir su impacto, aunque es casi imposible sacarlas por completo de nuestro código. Esto se hace más notable cuando se agregan nuevas características y el código va creciendo en tamaño y complejidad.

Errores como los Bugs de compilación pueden ser fáciles de detectar desde el principio, pero sin embargo otros como los errores en tiempo de ejecución son más problemáticos.

Los genéricos habilitan tipos (clases e interfaces) para ser parámetros al definir clases, interfaces y métodos. El código que utiliza genéricos tiene ventajas sobre el que no los usa: controles más fuertes en tiempo de compilación, eliminación de moldes, habilitación a los programadores para usar algoritmos genéricos.

Un tipo genérico es una clase genérica o interfaz que se ha parametrizado sobre tipos. Por convención, los nombres de los parámetros de tipo son letras individuales y en mayúscula, lo que ayuda a diferenciar entre una variable de tipo y el nombre de una clase o interfaz ordinaria.

Tipos raw pueden aparecer en el código heredado porque hay clases de API que no eran genéricos entes del JDK 5.0 y al utilizarlos se obtienen comportamientos pre-genéricos.

Cuando se mezcla código heredado con genérico, se pueden encontrar mensajes de advertencia, esto puede suceder cuando se utiliza un API más viejo que el que opera sobre el tipo raw.

El término "sin control" significa que el compilador no tiene suficiente información de tipo de realizar todas las revisiones de tipo necesarias para garantizar la seguridad de tipos.

Métodos genéricos son métodos que introducen sus propios parámetros de tipo. Esto es similar a la que se declara un tipo genérico, pero el alcance del parámetro de tipo es limitado al método donde se declara. La sintaxis de un método genérico incluye un parámetro de tipo, dentro de paréntesis angulares, y se presenta ante el tipo de retorno del método. Sin embargo la característica conocida como *inferencia de tipos*, permite invocar un método genérico como un método ordinario, sin especificar un tipo entre paréntesis angulares.

Los parámetros de tipo acotadas son clave para la implementación de algoritmos genéricos. Para declarar un parámetro de tipo acotado, escriba el nombre del parámetro de tipo, seguido por la palabra clave extendida, seguido de su límite superior. Tenga en cuenta que, en este contexto, extendida se utiliza en un sentido general para significar "extiende" (como en las clases) o "instrumentos" (como en las interfaces).

Es posible asignar un objeto de un tipo a un objeto de otro tipo, siempre y cuando los tipos sean compatibles. En terminología orientada a objetos se dice que es una relación. Lo mismo ocurre con los genéricos.

La inferencia de tipos es la capacidad de un compilador de Java para buscar en cada invocación del método y la declaración correspondiente para determinar el tipo de argumento (o argumentos) que hacen que la invocación aplicable. El algoritmo de inferencia determina los tipos de los argumentos y, en su caso, del tipo que se está asignando el resultado, o devuelto. Por último, el algoritmo de inferencia trata de encontrar el tipo *más específico* que trabaja con todos los argumentos.

La Inferencia de tipos y de instancias de clases genéricas puede reemplazar los argumentos de tipo necesarios para invocar el constructor de una clase genérica con un conjunto vacío de los parámetros de tipo (<>) siempre que el compilador pueda deducir los argumentos de tipo a partir del contexto.

En el código genérico, el signo de interrogación (?), llamado *comodín,* representa un tipo desconocido. El comodín se puede utilizar en una variedad de situaciones: como el tipo de un parámetro, un campo o variable local; a veces como un tipo de retorno.

Los genéricos se introdujeron al lenguaje Java para proporcionar controles más estrictos de tipo en tiempo de compilación y para apoyar la programación genérica.

Durante el proceso de borrado de tipo, el compilador Java borra todos los parámetros de tipo y reemplaza cada uno con su primera cota si el parámetro de tipo es limitado. Además el compilador Java también puede borrar los parámetros de tipo en los argumentos de los métodos genéricos. Pero hay que tener en cuenta que a veces el tipo de borrado provoca una situación que puede que no haya previsto. Al compilar una clase o interfaz que se extiende de una clase parametrizada o implementa una interfaz con parámetros, el compilador puede necesitar crear un método sintético, llamado *método de puente* (bridge), como parte del proceso de borrado tipo.

Un tipo *re-fiable* es un tipo cuya información de tipo es totalmente disponible en tiempo de ejecución. Esto incluye primitivos, tipos no genéricos, tipos raw y las invocaciones de comodines no consolidados. *Los tipos no re-fiable* son tipos en que la información se ha extraído en tiempo de compilación por borrado de tipo, este no tiene toda su información disponible en tiempo de ejecución.

Se produce mucha contaminación cuando una variable de un tipo parametrizado se refiere a un objeto que no es de ese tipo parametrizado. Esta situación se produce si el programa realiza alguna operación que da lugar a una advertencia sin control en tiempo de compilación.