EBook Gratis

APRENDIZAJE generics

Free unaffiliated eBook created from
Stack Overflow contributors.

#generics

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: generics

It is an unofficial and free generics ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official generics.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con los genéricos

Observaciones

Los genéricos le permiten definir marcadores de posición para tipos exactos en definiciones para clases, interfaces y / o métodos.

Subtemas posibles:

- Clase (y estructura) genéricos
- Genéricos de interfaz
- Método genérico
- Restricciones
- Covarianza y contravarianza.

Examples

Disponibilidad

Los genéricos estuvieron disponibles con:

- .NET Framework 2.0 (y la versión 2.0 del framework compacto).
- Java en la versión 5.
- Common Lisp ya que estaba estandarizado ...

Lea Empezando con los genéricos en línea:
https://riptutorial.com/es/generics/topic/4454/empezando-con-los-genericos

Capítulo 2: Genéricos en Java

Sintaxis

- `class MyClass<T1, T2 extends CharSequence> implements Comparable<MyClass> //...`
- `interface MyListInterface<T extends Serializable> extends List<T> //...`
- `public <T1, T2 extends Instant> T1 provideClone(T1 toClone, T2 instant> //...`
- `public static List<CharSequence> safe(Collection<? extends CharSequence> l) { return new ArrayList<>(l);}`
- `Set<String> strings = Collections.singleton("Hello world");`
- `List<CharSequence> chsList = safe(strings);`

Observaciones

El borrado de tipos limita la reflexión, aunque eso no es específico de JVM, por ejemplo, Ceilán usa genéricos reificados .

El soporte de tipo existencial no es necesariamente compatible con otros idiomas de esta forma: Kotlin lo soporta a través de proyecciones de tipo .

Examples

Introducción

Los genéricos se introdujeron en Java en su versión (1) 5. Estos se borran durante la compilación, por lo que la reflexión en tiempo de ejecución no es posible para ellos. Los genéricos generan nuevos tipos parametrizados por otros tipos. Por ejemplo, no tenemos que crear nuevas clases para usar la colección segura de tipos de `String` sy `Number` s, `ArrayList<T>` genérico `ArrayList<T>` se puede usar en todos los casos, como: `new ArrayList<String>()` .

Ejemplo:

```
List<String> variable = new ArrayList<String>();
```

En Java 7, se introdujo algo de azúcar sintáctico para facilitar la construcción (`<>` aka. Diamante):

```
List<String> variable = new ArrayList<>();
```

Curiosamente, también fue posible (desde Java 5) usar la inferencia de tipos, cuando un método estático tenía un valor de retorno (a menudo utilizado en Google Guava, por ejemplo):

```
List<String> singleton = Collections.singletonList();//Note the missing `<>` or `<String>`!
```

En Java, los tipos existenciales se usaron para proporcionar polimorfismo para los tipos, ya que los tipos genéricos son invariantes (por ejemplo: `List<String>` no es un subtipo, ni un supertipo de `List<CharSequence>` , aunque en Java `String[]` es un subtipo de `CharSequence[]` ; nota: `String`

implementa la interfaz `CharSequence`). Los tipos genéricos existenciales se pueden expresar como:

```
List<? extends CharSequence> list = new ArrayList<String>();
Comparable<? super ChronoLocalDate> ccld = LocalDate.now();
ChronoLocalDate cld = JapaneseDate.now(); //ChronoLocalDate extends
Comparable<ChronoLocalDate>
ccld.compareTo(cld);
//cld.compareTo(ccld);//fails to compile because ccld is not a `ChronoLocalDate` (compile
time)
```

Ambas instancias se pueden usar en una lista parametrizada por el `Comparable` correspondiente:

```
List<Comparable<? super ChronoLocalDate>> list2 = new ArrayList<>();
list2.add(cld);
list2.add(ccld);
```

Métodos genéricos

Los *parámetros de tipo* genérico se definen comúnmente a nivel de clase o interfaz, pero los *métodos* y (rara vez) los *constructores* también admiten la declaración de parámetros de tipo vinculados al alcance de una sola llamada de método.

```
class Utility // no generics at the class level
{
    @SafeVarargs
    public static <T> T randomOf(T first, T... rest) {
        int choice = new java.util.Random().nextInt(rest.length + 1);
        return choice == rest.length ? first : rest[choice];
    }

    public static <T extends Comparable<T>> T max(T t1, T t2) {
        return t1.compareTo(t2) < 0 ? t2 : t1;
    }
}
```

Observe que las declaraciones de parámetros de tipo, `T` y `<T extends Comparable<T>>` respectivamente, aparecen después de los modificadores de método y *antes* del tipo de retorno. Esto permite que el parámetro de tipo `T` se use dentro del alcance de tales métodos, actuando como:

- tipos de argumento
- tipo de retorno
- tipos de variables locales

Aunque los dos métodos anteriores utilizan el mismo nombre de parámetro de tipo `T` , en el nivel de método son completamente independientes entre sí. El compilador *inferirá* el tipo real basado en los argumentos pasados al método *en cada sitio de llamada* que invoque el método. Dado que el método `max` declara que `T extends Comparable<T>` , el compilador también impone que los tipos inferidos sean implementaciones compatibles de la interfaz `Comparable` .

```
Integer num1 = 1;
```

```
Integer num2 = 2;
String str1 = "abc";
String str2 = "xyz";

Integer bigger = Utility.max(num1, num2);
assert bigger == num2;

String later = Utility.max(str2, str1);
assert later == str2;

Utility.max(num1, str1); // compiler error: num1 and str1 are incompatible types

Utility.max(new Object(), new Object()); // compiler error: Object does not implement
Comparable
```

Java 8 mejoró significativamente la capacidad del compilador para inferir correctamente los tipos genéricos en los sitios de llamadas. Si el compilador no puede inferir el tipo adecuado, los desarrolladores pueden declarar explícitamente el tipo como parte de la llamada:

```
Object obj = Utility.<Object>randomOf(str1, new Object(), num1);
```

Lea Genéricos en Java en línea: https://riptutorial.com/es/generics/topic/6552/genericos-en-java

Creditos

S. No	Capítulos	Contributors
1	Empezando con los genéricos	Community, Mark Hurd
2	Genéricos en Java	Gábor Bakos, William Price