Generics

Contenido

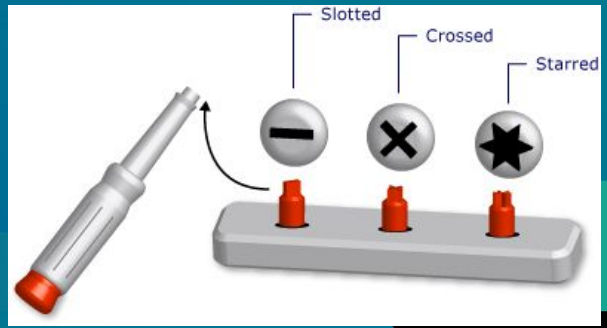
Generics

- ¿Qué son los genéricos?
- Beneficios.
- Sintaxis básica.
- Parámetros de tipos limitados.
- Wildcards.
- Limitaciones de los genéricos.
- Parámetros de tipos limitados múltiples.

¿Qué son los genéricos?

Los **generics** son una característica que permite definir *clases*, *interfaces* y *métodos* con tipos de datos que se especifican al momento de su uso, en lugar de en su declaración.

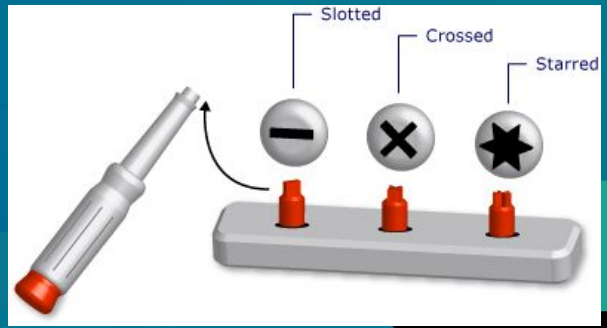
Esto ofrece mayor seguridad y flexibilidad en el código, evitando la necesidad de hacer casting y permitiendo detectar errores en tiempo de compilación.



¿Qué son los genéricos?

Los **genéricos** ayudan a evitar errores en tiempo de ejecución y a tener un código más seguro y limpio al restringir los tipos que se pueden almacenar en colecciones.

Antes de los genéricos, todas las colecciones de Java aceptaban objetos de cualquier tipo, lo que requería un **casting explícito** al obtener los elementos, lo que era propenso a errores.

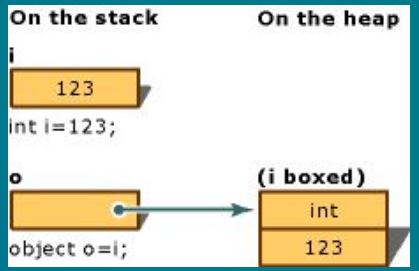


Beneficios

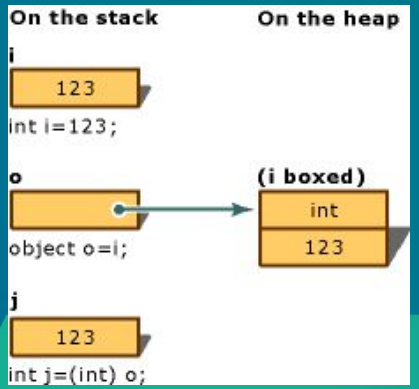
SEGURIDAD DE TIPOS
- Previene posibles errores.

REUTILIZACIÓN DE CÓDIGO
- Mismas tareas con distintos tipos.

GENERICS vs NO GENERICS
- Eficiencia. Evita el boxing / unboxing.



BOXING



UNBOXING

Sintaxis básica

La sintaxis general de un tipo genérico es **<T>**, donde **T** es un *parámetro de tipo* que puede ser cualquier nombre de letra (por convención, se suelen usar letras como T, U, V). Los genéricos se pueden aplicar a:

- **Clases**:

```
public class Generica<T> { ... }
```

- **Interfaces**:

```
public interface Comparable<T> { ... }
```

- **Métodos**:

```
public <T> void metodo(T item) { ... }
```

Parámetros de tipo limitados

Es posible restringir los tipos que puedan aceptar un parámetro genérico utilizando la palabra reservada **extends**.

Límite superior (extends): Se utiliza para restringir un parámetro de tipo a una clase específica o cualquiera de sus subclases (solo lectura).

```
public <T extends Number> void mostrarNumero(T value) { ... }
```

```
public <T extends Persona> void mostrarPersona(T value) { ... }
```

```
public class Numerica<T extends Number> { ... }
```

```
public class Terricola<T extends Persona> { ... }
```

Wildcards

A veces, no es necesario especificar un tipo exacto, sino un "comodín" que permita varios tipos relacionados.

En estos casos, se puede usar **?**, que representa un *tipo desconocido*.

- **Wildcard sin restricciones**: `List<?>`
  - acepta cualquier tipo de lista.
- **Wildcard limitado superior**: `List<? extends Number>`
  - acepta cualquier lista cuyos elementos son de un tipo que extiende Number.
- **Wildcard limitado inferior**: `List<? super Integer>`
  - acepta listas de Integer o de cualquiera de sus superclases, como Number u Object.

Limitaciones de los genéricos

- **No se puede instanciar un tipo genérico**: `T obj = new T(); //no es válido.`
- **No se puede usar tipos primitivos** directamente como parámetros de tipo (int, double, etc.).
  En su lugar, se deben usar las clases *wrapper* (*Integer*, *Double*, *etc.*).
- **Los genéricos no existen en tiempo de ejecución** debido al "borrado de tipos" (*type erasure*) de Java.
  Esto significa que el tipo genérico se elimina en tiempo de ejecución y se convierte en Object, lo que impone ciertas restricciones.
  Por ejemplo no se pueden sobrecargar métodos usando diferentes tipos genéricos, ya que en tiempo de ejecución el tipo genérico se considera el mismo.

```
public void metodo(Generica<Integer> g) { ... }
public void metodo(Generica<String> g) { ... }// ERROR: Se considera la misma firma
```

Parámetros de tipo limitados múltiples

Además de un límite inferior o superior, se pueden especificar parámetros genéricos que implementen ciertas interfaces.

```java
public <T extends Number & Comparable<T>> T encontrarMaximo(T a, T b) {
    return (a.compareTo(b) > 0) ? a : b;
}
```

```java
public interface Identificable {
    String getDNI();
}

public class Persona implements Identificable, Comparable<Persona> { ... }
```

```java
public static <T extends Persona & Identificable> void mostrarDNI(T obj) {
    System.out.println("ID: " + obj.getDNI());
}
```

Ejercitación