Genericos en Java

...

Programación II , en base a https://dev.java/learn/generics/intro/

Principio DRY

El principio DRY (Don't Repeat Yourself) es un importante en programación. Fue acuñado por Andy Hunt y Dave Thomas en su libro "The Pragmatic Programmer" (El programador pragmático). El principio DRY se enfoca en la eliminación de la duplicación de código en un sistema de software.

El principio DRY establece que cada pieza de conocimiento o lógica en un programa debe tener una única representación autoritaria y clara en todo el sistema. En lugar de duplicar el mismo código en múltiples lugares, se debe crear una abstracción o función reutilizable que encapsule ese comportamiento repetido. De esta manera, si surge la necesidad de realizar cambios, solo se requiere modificar un único lugar en lugar de múltiples puntos dispersos en el código.

Definición

En pocas palabras, los genéricos permiten que los tipos (clases e interfaces) sean parámetros al definir clases, interfaces y métodos. Al igual que los parámetros formales más familiares utilizados en las declaraciones de métodos, los parámetros de tipo proporcionan una manera de reutilizar el mismo código con diferentes entradas. La diferencia es que las entradas a los parámetros formales son valores, mientras que las entradas a los parámetros de tipo son tipos.

Ventajas

- Comprobaciones de tipos más estrictas en tiempo de compilación. Un compilador de Java aplica una verificación de tipos estricta al código genérico y emite errores si el código viola la seguridad de tipos. Corregir errores en tiempo de compilación es más fácil que corregir errores en tiempo de ejecución, que pueden ser difíciles de encontrar.
- Permitir a los programadores implementar algoritmos genéricos. Al utilizar genéricos, los programadores pueden implementar algoritmos genéricos que funcionan en colecciones de diferentes tipos, se pueden personalizar y son seguros y fáciles de leer.

Problema

```
List list = new ArrayList();

list.add("hello");

String s = (String) list.get(0);
```

Solución al problema de la diapositiva anterior

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

list.add("hello");

String s = list.get(0);   // no cast
```

Clase de ejemplo

```
public class DesdeHasta <T,U>{
	private T desde;
	private U hasta;
	public DesdeHasta() {
	}
	public T getDesde() {
		return desde;
	}
	public void setDesde(T desde) {
		this.desde = desde;
	}
	// getters and setters .....
}
```

Ejemplo de uso

```
DesdeHasta<Integer,Integer> desdeHasta = new DesdeHasta<Integer, Integer>();

desdeHasta.setDesde(100);

desdeHasta.setHasta(200);

System.out.println(desdeHasta);
```

Ejercicio 1

Modelar un sistema que tenga un nombre y una lista de clientes. Permitiendo agregar clientes a la lista. De los clientes se registra el dni y el apellido. El dni no puede estar vacío. La lista tiene hasta 10 integrantes. Se pide realizar los TAD, IREP e implementación en java.

Ejercicio

Se requiere modelar un campeonato de carreras. Se pide ingresar nombre de corredor, posición, tiempo empleado y categoría. Es obligatorio el nombre del corredor. Existen tres categorías: senior: hasta 500 participantes, junior: hasta 1000 participantes, professional: 100 participantes. Se pide dejar preparado para agregar más categorías y categorías numéricas.. Listar los primeros 3 participantes de cada categoría según los datos cargados en posición. Realizar los TAD e invariantes de representación. Bonus: A cada participante junior, se registra un premio, para los 10 primeros. Para los seniors, se entrega una medalla y para los profesionales, para los primeros 3, medalla de oro, bronce y cobre. Informar en cada caso.