**Generics**

Os **generics** em Java são um recurso que permite criar classes, interfaces e métodos parametrizados por tipo. Isso significa que podemos especificar o tipo de dados que será usado sem precisar definir explicitamente o tipo na implementação, aumentando a flexibilidade e a reutilização do código. Vamos explorar os benefícios mencionados:

**1. Reuso**

Os generics permitem criar classes, interfaces ou métodos que podem trabalhar com diferentes tipos de dados sem precisar reescrevê-los.

**Exemplo:**

public class Box<T> {

private T content;

public void setContent(T content) {

this.content = content;

}

public T getContent() {

return content;

}

}

// Uso:

Box<String> stringBox = new Box<>();

stringBox.setContent("Hello");

Box<Integer> integerBox = new Box<>();

integerBox.setContent(123);

A classe Box pode ser usada com qualquer tipo (String, Integer, etc.), eliminando a necessidade de criar várias versões específicas.

**2. Type Safety**

Com generics, o compilador verifica o tipo em tempo de compilação, reduzindo erros de tipo em tempo de execução. Isso melhora a segurança do código.

**Sem generics (antes de Java 5):**

List list = new ArrayList();

list.add("Hello");

String item = (String) list.get(0); // Necessário cast

**Com generics:**

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

list.add("Hello");

String item = list.get(0); // Não é necessário cast

No exemplo com generics, o compilador garante que apenas strings podem ser adicionadas à lista.

**3. Performance**

Embora os generics sejam implementados por meio de **type erasure** (remoção do tipo parametrizado em tempo de compilação), eles eliminam a necessidade de conversões explícitas de tipo (casting), o que pode melhorar a performance em certos cenários.

**Sem generics:**

List list = new ArrayList();

list.add(10); // Autoboxing de int para Integer

int number = (Integer) list.get(0); // Necessário cast

Com generics:

List<Integer> list = new ArrayList<>();

list.add(10); // Autoboxing ocorre, mas sem necessidade de cast ao recuperar

int number = list.get(0);

A eliminação de casts reduz o custo computacional e melhora a legibilidade.

**4. Uso comum: Coleções**

Os generics são amplamente usados em coleções da API do Java (java.util), como List, Set, Map, e outros.

**Exemplo com List:**

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

for (String name : names) {

System.out.println(name);

}

Exemplo com Map:

Map<Integer, String> map = new HashMap<>();

map.put(1, "One");

map.put(2, "Two");

for (Map.Entry<Integer, String> entry : map.entrySet()) {

System.out.println(entry.getKey() + " -> " + entry.getValue());

}

**Conclusão**

Os generics trazem uma série de vantagens, como:

* **Reuso:** Criação de código reutilizável.
* **Type Safety:** Garantia de segurança de tipo em tempo de compilação.
* **Performance:** Eliminação de casts desnecessários.
* **Facilidade no uso de coleções:** Simplifica o uso de estruturas como listas e mapas.

Isso resulta em código mais limpo, seguro e eficiente.