POO – JAVA 21

PROGRAMAÇÃO ORIENTADA A OBJETO

Prof. Alex Gondim Lima

INSTITUTO FEDERAL
Bahia
Campus
Santo Antônio
de Jesus

INSTITUTO FEDERAL
Bahia
Campus
Santo Antônio
de Jesus

Apresentação

- Contato : 75 999331334
- Alex Gondim Lima
- agllima@ufba.br

O que são Generics?

- Generics (ou “tipos genéricos”) foram introduzidos no Java 5.
- Permitem criar classes, interfaces e métodos que podem receber tipos como parâmetro.
- Tornam o código mais seguro (type-safety) e reduzem a necessidade de casts.

Vantagens dos Generics

1. **Type-safety**: Evita erros de tipo em tempo de execução.
2. **Reutilização de Código**: Uma mesma classe/método pode trabalhar com diferentes tipos.
3. **Redução de Casts**: Não precisamos mais converter tipos ao obter objetos da coleção.
4. **Legibilidade**: O código fica mais claro sobre qual tipo está sendo usado.

Exemplo simples de Classe Genérica

```
public class Caixa<T> {
    private T objeto;

    public void setObjeto(T objeto) {
        this.objeto = objeto;
    }

    public T getObjeto() {
        return objeto;
    }
}
```

Exemplo de Classe Genérica

Classe Caixa usada com diferentes tipos:

```
public class Main {
    public static void main(String[] args) {
        Caixa<String> caixaString = new Caixa<>();
        caixaString.setObjeto("Olá, Generics!");
        String texto = caixaString.getObjeto();

        Caixa<Integer> caixaInt = new Caixa<>();
        caixaInt.setObjeto(100);
        Integer numero = caixaInt.getObjeto();
    }
}
```

Generics em Coleções
Exemplo com List<T>:

```
List<String> listaNomes = new ArrayList<>();
listaNomes.add("Ana");
listaNomes.add("Carlos");

for (String nome : listaNomes) {
    System.out.println(nome);
}
```

Métodos Genéricos
Definição de método genérico:

```
public static <T> void imprimirArray(T[] array) {
    for (T elem : array) {
        System.out.println(elem);
    }
}
```

Exemplo de Método Genérico

```java
public class Main {
    public static void main(String[] args) {
        String[] nomes = {"Ana", "Carlos", "Bruna"};
        Integer[] numeros = {1, 2, 3, 4};

        imprimirArray(nomes);
        imprimirArray(numeros);
    }

    public static <T> void imprimirArray(T[] array) {
        for (T elem : array) {
            System.out.print(elem + " ");
        }
        System.out.println();
    }
}
```

Diamond Operator (<>)

•Desde o Java 7, não precisamos repetir o tipo nos construtores de coleções.

•Em vez de List<String> lista = new ArrayList<String>(); podemos usar

List<String> lista = new ArrayList<>();
•O compilador infere o tipo automaticamente.

Wildcards (Curingas)

1.? = “desconhecido”

- Usado quando não sabemos o tipo exato.

2.? extends T = qualquer classe que herde de T.

- Principalmente para ler dados de uma estrutura.

3.? super T = qualquer superclasse de T.

- Principalmente para escrever dados em uma estrutura.

Exemplo de Wildcard Simples

```java
public static void imprimirLista(List<?> lista) {
    for (Object elem : lista) {
        System.out.println(elem);
    }
}

public static void main(String[] args) {
    List<String> palavras = Arrays.asList("Java", "Generics");
    List<Integer> numeros = Arrays.asList(1, 2, 3);

    imprimirLista(palavras);
    imprimirLista(numeros);
}
```

Boas Práticas

1. **Sempre usar Generics em coleções**: Evita casts e erros de tipo.
2. **Preferir Interfaces**: Declare List<String> lista = new ArrayList<>(); em vez de usar a classe concreta no tipo.
3. **Utilizar o operador diamante**: Simplifica a declaração.
4. **Não ignorar Warnings**: Unchecked warnings podem indicar problemas sérios de tipo.
5. **Projetar classes e métodos genéricos**: Se frequentemente criamos código duplicado para diferentes tipos, considere torná-lo genérico.

Conclusão

•Generics são uma **ferramenta essencial** no desenvolvimento moderno em Java, garantindo segurança de tipo e reduzindo erros de casting.

•Facilitam a **manutenção** e **evolução** do código, além de melhorarem a legibilidade.

•Aprofundar-se em **wildcards** e **bounded types** (extends/super) é fundamental para aproveitar todo o potencial de Generics.