Generics em Java

✔ Explicações teóricas
✔ Vantagens e desvantagens
✔ Onde e quando usar
✔ 10 exemplos práticos de mercado
✔ Cada exemplo contém:

- Descrição
- Código em Java
- Exemplo de uso real
- Explicação
  ✔ CRUD Genérico completo (sem JPA)

📘 AULA COMPLETA – GENERICS EM JAVA (1 HORA)

Material pronto para você usar em aula, slides, videoaula ou apostila.

🚀 1. O que são Generics?

Generics são um recurso do Java que permite criar **classes, métodos e interfaces que trabalham com tipos parametrizados** — ou seja, você define **um tipo como parâmetro**, aumentando segurança, reutilização e clareza.

Exemplo simples:

```
java

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

O `String` é o *tipo genérico* dessa lista.

⭐ 2. Vantagens dos Generics

✔ 2.1. Segurança de tipo (type safety)

Você evita erros de cast em tempo de execução.

✔️ 2.2. Menos código repetido

É possível criar componentes reutilizáveis como repositórios, serviços e utilitários genéricos.

✔️ 2.3. Melhor legibilidade e manutenção

O código fica mais claro, com tipos explícitos.

✔️ 2.4. Melhora a produtividade

Menos duplicação e menos bugs.

⚠️ 3. Desvantagens dos Generics

❌ Não funciona bem com tipos primitivos (usa wrappers)

Ex: `List<int>` não existe → usar `List<Integer>`.

❌ Erasure (apagamento de tipo)

O Java apaga o tipo genérico em tempo de execução, limitando certas operações.

❌ Complexidade maior em casos avançados

Bounded types, wildcards e generics aninhados podem ficar difíceis para iniciantes.

🎯 4. Onde e Quando Usar Generics

- ✔ Criar repositórios reutilizáveis
- ✔ Criar serviços que lidam com múltiplos tipos
- ✔ Implementar filtros, validadores e componentes de alto nível
- ✔ Evitar duplicação de código
- ✔ Trabalhar com listas, maps e coleções
- ✔ Criar API's reutilizáveis
- ✔ Padronizar respostas de APIs (Response<T>)
- ✔ Estruturas de dados
- ✔ Padrões Repository, Factory e Strategy

🔟 5. 10 Exemplos Práticos de Mercado (com código, uso e explicação)

✅ Exemplo 1 — Validador Genérico

(Validator<T>)

Descrição

Sistema que valida vários tipos diferentes sem duplicar código.

Código

```
java

public interface Validator<T> {
    boolean isValid(T value);
}

public class EmailValidator implements Validator<String> {
    @Override
    public boolean isValid(String email) {
        return email.contains("@");
    }
}

public class NumberValidator implements Validator<Integer> {
    @Override
    public boolean isValid(Integer number) {
        return number > 0;
    }
}
```

Exemplo de Uso

```
java

Validator<String> emailVal = new EmailValidator();
System.out.println(emailVal.isValid("alex@email.com")); // true

Validator<Integer> numberVal = new NumberValidator();
System.out.println(numberVal.isValid(10)); // true
```

Explicação

Você cria um validador genérico que pode validar qualquer tipo, aplicando regras diferentes sem duplicar código.

✅ Exemplo 2 — Service de Resposta Genérica (Response<T>)

Descrição

Muito usado em APIs REST.

Código

```
java

public class Response<T> {
    private T data;
    private String message;

    public Response(T data, String message) {
        this.data = data;
        this.message = message;
    }

    public T getData() { return data; }
    public String getMessage() { return message; }
}
```

Uso

```
java

Response<String> r1 = new Response<>("OK", "Operação realizada");
Response<Integer> r2 = new Response<>(200, "Status code");

System.out.println(r1.getData());
System.out.println(r2.getData());
```

Explicação

Permite retornar **qualquer tipo** sem criar múltiplas classes de resposta.

✅ Exemplo 3 — Filtro Genérico (Filter<T>)

Descrição

Criar filtros reutilizáveis para listas de objetos.

Código

```java
public interface Filter<T> {
    boolean apply(T item);
}

public class StartsWithFilter implements Filter<String> {
    private String prefix;

    public StartsWithFilter(String prefix) {
        this.prefix = prefix;
    }

    public boolean apply(String item) {
        return item.startsWith(prefix);
    }
}
```

Uso

```java
Filter<String> filter = new StartsWithFilter("A");
System.out.println(filter.apply("Alex")); // true
```

Explicação

Padrão muito usado em buscas avançadas.

✅ Exemplo 4 — Repositório em Memória Genérico (InMemoryRepository<T, ID>)

Descrição

Base para repositórios reutilizáveis.

Código

```java
public class InMemoryRepository<T, ID> {

    private Map<ID, T> storage = new HashMap<>();

    public void save(ID id, T entity) {
        storage.put(id, entity);
    }

    public T findById(ID id) {
        return storage.get(id);
    }

    public List<T> findAll() {
        return new ArrayList<>(storage.values());
    }

    public void delete(ID id) {
        storage.remove(id);
    }
}
```

Uso

```java
InMemoryRepository<String, Integer> repo = new InMemoryRepository<>
();
repo.save(1, "Alex");
System.out.println(repo.findById(1));
```

Explicação

Um CRUD simples e reutilizável.

✅ Exemplo 5 — Classe Utilitária Genérica (Pair<T, U>)

Código

```java
public class Pair<T, U> {
    private T first;
    private U second;

    public Pair(T first, U second) {
        this.first = first;
        this.second = second;
    }

    public T getFirst() { return first; }
    public U getSecond() { return second; }
}
```

Uso

```java
Pair<String, Integer> user = new Pair<>("Alex", 34);
```

Explicação

Estrutura útil em diversos cenários.

✅ Exemplo 6 — Cache Genérico (Cache<K, V>)

```java
public class Cache<K, V> {
    private Map<K, V> cache = new HashMap<>();

    public void put(K key, V value) {
        cache.put(key, value);
    }

    public V get(K key) {
        return cache.get(key);
    }
}
```

Uso

```java
Cache<String, String> cache = new Cache<>();
cache.put("token", "ABC");
```

✅ Exemplo 7 — Conversor Genérico (Converter<T, R>)

```java
public interface Converter<T, R> {
    R convert(T source);
}

public class UserToDTO implements Converter<User, UserDTO> {
    @Override
    public UserDTO convert(User user) {
        return new UserDTO(user.getNome(), user.getEmail());
    }
}
```

Uso

```
java

Converter<User, UserDTO> conv = new UserToDTO();
```

✅ Exemplo 8 — Factory Genérica (Factory<T>)

```
java

public interface Factory<T> {
    T create();
}

public class UserFactory implements Factory<User> {
    public User create() {
        return new User("Novo Usuário");
    }
}
```

✅ Exemplo 9 — Ordenação Genérica (Sorter<T>)

```
java

public class Sorter<T extends Comparable<T>> {
    public List<T> sort(List<T> list) {
        Collections.sort(list);
        return list;
    }
}
```

✅ Exemplo 10 — Builder Genérico (Builder<T>)

```java
public interface Builder<T> {
    T build();
}

public class UserBuilder implements Builder<User> {
    private String nome;

    public UserBuilder setNome(String nome) {
        this.nome = nome;
        return this;
    }

    public User build() {
        return new User(nome);
    }
}
```

🔥 6. CRUD GENÉRICO COMPLETO EM MEMÓRIA (SEM JPA)

Perfeito para usar em projetos pequenos, testes ou aulas.

🧱 Modelo

```java
public interface GenericRepository<T, ID> {
    void save(ID id, T entity);
    T findById(ID id);
    List<T> findAll();
    void update(ID id, T entity);
    void delete(ID id);
}
```

Implementação

java

```
public class GenericRepositoryImpl<T, ID> implements
GenericRepository<T, ID> {

    private Map<ID, T> data = new HashMap<>();

    @Override
    public void save(ID id, T entity) {
        data.put(id, entity);
    }

    @Override
    public T findById(ID id) {
        return data.get(id);
    }

    @Override
    public List<T> findAll() {
        return new ArrayList<>(data.values());
    }

    @Override
    public void update(ID id, T entity) {
        data.put(id, entity);
    }

    @Override
    public void delete(ID id) {
        data.remove(id);
    }
}
```

👤 Entidade de Exemplo

```java
public class User {
    private Integer id;
    private String nome;

    public User(Integer id, String nome) {
        this.id = id;
        this.nome = nome;
    }
}
```

💻 Uso do CRUD

```java
public class TestCRUD {
    public static void main(String[] args) {

        GenericRepository<User, Integer> repo = new
GenericRepositoryImpl<>();

        repo.save(1, new User(1, "Alex"));
        repo.save(2, new User(2, "João"));

        System.out.println(repo.findById(1).getNome());

        repo.update(1, new User(1, "Alex Fernando"));

        repo.delete(2);

        repo.findAll().forEach(u ->
System.out.println(u.getNome()));
    }
}
```