Conteúdo sobre Reflection

🔍 O que é Reflection em Java (conceito e como funciona)

Reflection é a capacidade que o Java oferece para **inspecionar e manipular classes, métodos, campos e construtores em tempo de execução**, mesmo que eles sejam **privados** ou **desconhecidos em tempo de compilação**.

Ela funciona através da **Reflection API**, principalmente no pacote:

```
java.lang.reflect
```

Como funciona na prática

1. Você obtém um objeto `Class`
2. A partir dele, acessa `Method`, `Field`, `Constructor`
3. Executa operações dinamicamente

Exemplo curto

```
Class<?> clazz = Class.forName("com.exemplo.Usuario");
Method metodo = clazz.getMethod("getNome");
Object obj = clazz.getDeclaredConstructor().newInstance();

String nome = (String) metodo.invoke(obj);
```

➡️ O método **não foi chamado diretamente**, mas **descoberto e executado em runtime**.

✅ Vantagens e benefícios reais

1️⃣ Flexibilidade extrema

Permite trabalhar com classes **desconhecidas em tempo de compilação**.

> Ideal para frameworks, plugins, engines e ferramentas genéricas.

2️⃣ Base de frameworks modernos

Frameworks como **Spring Framework**, **Hibernate** e **Jackson** usam Reflection intensivamente.

3️⃣ Redução de acoplamento

Você não depende diretamente da implementação concreta.

4️⃣ Automação e metaprogramação

Geração dinâmica de:

- Mapeamentos
- Injeções
- Serializações
- Validações

⚠️ Desvantagens e riscos

❌ Performance

Reflection é **mais lenta** que chamadas diretas.

```
metodo.invoke(obj); // mais lento que obj.metodo()
```

❌ Quebra de encapsulamento

```
field.setAccessible(true);
```

➡️ Viola princípios de **POO**.

❌ Segurança

Pode acessar dados sensíveis indevidamente.

❌ Manutenção difícil

- Erros só aparecem em runtime

- Refatorações quebram código silenciosamente

❓ Por que usar Reflection

Use Reflection quando:

- Você **não sabe qual classe será usada**
- Precisa criar **infraestrutura genérica**
- Está desenvolvendo **frameworks, SDKs ou engines**

⏰ Quando usar (casos recomendados)

✔️ Casos ideais

- Injeção de dependência (DI)
- Serialização / desserialização
- Processamento de anotações
- Testes automatizados
- Plugins dinâmicos
- Mapeamento ORM

Exemplo curto – leitura de anotação

```
if (classe.isAnnotationPresent(Entity.class)) {
    System.out.println("É uma entidade JPA");
}
```

🚫 Quando NÃO usar (anti-patterns)

❌ Anti-patterns comuns

- Usar Reflection para chamar métodos conhecidos
- Substituir polimorfismo por Reflection
- Usar Reflection em código de negócio

- Usar Reflection em loops críticos de performance

❌ Exemplo ruim

```
method.invoke(obj); // quando você conhece o método
```

🏢 Onde Reflection é usada no mercado

Exemplos reais

- **Spring Boot** → DI, proxies, beans
- **Hibernate** → mapeamento ORM
- **Jackson** → JSON ↔ Objeto
- **JUnit** → execução dinâmica de testes
- Plugins (IDE, servidores, engines)

🧠 Boas práticas e cuidados

✔️ Boas práticas

1️⃣ Cache de objetos Reflection

```
static final Method METODO = Classe.class.getMethod("getNome");
```

2️⃣ Use `setAccessible(true)` com cuidado

Somente quando **realmente necessário**.

3️⃣ Trate exceções corretamente

```
java

try {
    method.invoke(obj);
} catch (ReflectiveOperationException e) {
    log.error("Erro em reflection", e);
}
```

4️⃣ Nunca exponha Reflection ao usuário final

Use internamente na infraestrutura.

5️⃣ Documente MUITO bem

🔹 Exemplo 01 – Descobrir informações de uma classe (Framework / Debug)

📌 Uso real

Frameworks inspecionam classes para entender sua estrutura.

Classe `Usuario`

```
public class Usuario {
    private Long id;
    private String nome;

    public void login() {}
    public void logout() {}
}
```

Main

```
import java.lang.reflect.*;

public class Main {
    public static void main(String[] args) {
        Class<Usuario> clazz = Usuario.class;

        System.out.println("Campos:");
        for (Field f : clazz.getDeclaredFields()) {
            System.out.println("- " + f.getName());
        }

        System.out.println("\nMétodos:");
        for (Method m : clazz.getDeclaredMethods()) {
            System.out.println("- " + m.getName());
        }
    }
}
```

🔹 Exemplo 02 – Executar método dinamicamente (Plugin / Engine)

📌 Uso real

Plugins carregados dinamicamente.

Classe `ServicoEmail`

```
public class ServicoEmail {
    public void enviar() {
        System.out.println("Email enviado!");
    }
}
```

Main

```
import java.lang.reflect.Method;

public class Main {
    public static void main(String[] args) throws Exception {
        Class<?> clazz = Class.forName("ServicoEmail");
        Object obj = clazz.getDeclaredConstructor().newInstance();

        Method method = clazz.getMethod("enviar");
        method.invoke(obj);
    }
}
```

🔹 Exemplo 03 – Acessar atributo privado (Framework / ORM)

📌 Uso real

Hibernate, Jackson, Spring.

Classe `Produto`

```
java

public class Produto {
    private String nome = "Notebook";
}
```

Main

```
import java.lang.reflect.Field;

public class Main {
    public static void main(String[] args) throws Exception {
        Produto produto = new Produto();

        Field field = Produto.class.getDeclaredField("nome");
        field.setAccessible(true);

        System.out.println(field.get(produto));
    }
}
```

🔹 Exemplo 04 – Alterar campo privado (Injeção de dependência)

📌 Uso real

Spring DI.

Classe `Config`

```
public class Config {
    private String url;
}
```

Main

```
import java.lang.reflect.Field;

public class Main {
    public static void main(String[] args) throws Exception {
        Config config = new Config();

        Field field = Config.class.getDeclaredField("url");
        field.setAccessible(true);
        field.set(config, "jdbc:postgresql://localhost/db");

        System.out.println("URL configurada com sucesso");
    }
}
```

🔹 Exemplo 05 – Instanciar classe pelo nome (Factory dinâmica)

📌 Uso real

Factories genéricas.

Classe `ServicoPagamento`

```
public class ServicoPagamento {
    public ServicoPagamento() {
        System.out.println("Serviço de pagamento criado");
    }
}
```

Main

```
public class Main {
    public static void main(String[] args) throws Exception {
        Class<?> clazz = Class.forName("ServicoPagamento");
        Object obj = clazz.getDeclaredConstructor().newInstance();
    }
}
```

🔹 Exemplo 06 – Executar métodos anotados (Framework)

📌 Uso real

JUnit, Spring.

Anotação

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Executar {}
```

Classe

```
public class Tarefa {
    @Executar
    public void rodar() {
        System.out.println("Tarefa executada");
    }
}
```

Main

```
import java.lang.reflect.Method;

public class Main {
    public static void main(String[] args) throws Exception {
        Tarefa tarefa = new Tarefa();

        for (Method m : tarefa.getClass().getDeclaredMethods()) {
            if (m.isAnnotationPresent(Executar.class)) {
                m.invoke(tarefa);
            }
        }
    }
}
```

🔹 Exemplo 07 – Descobrir construtores (Framework / Container)

📌 Uso real

Containers IoC.

Classe

```
public class Cliente {
    public Cliente() {}
    public Cliente(String nome) {}
}
```

Main

```
import java.lang.reflect.Constructor;

public class Main {
    public static void main(String[] args) {
        for (Constructor<?> c :
Cliente.class.getDeclaredConstructors()) {
            System.out.println(c);
        }
    }
}
```

🔹 Exemplo 08 – Validação genérica de campos (Framework)

📌 Uso real

Validação automática.

Classe

```
public class Usuario {
    private String nome;
}
```

Main

```
import java.lang.reflect.Field;

public class Main {
    public static void main(String[] args) throws Exception {
        Usuario u = new Usuario();

        for (Field f : u.getClass().getDeclaredFields()) {
            f.setAccessible(true);
            if (f.get(u) == null) {
                System.out.println("Campo " + f.getName() + " está
nulo");
            }
        }
    }
}
```

🔹 Exemplo 09 – Mapper simples (DTO ↔ Entidade)

📌 Uso real

MapStruct, ModelMapper.

Classes

```
public class UsuarioDTO {
    public String nome;
}

public class Usuario {
    public String nome;
}
```

Main

```
import java.lang.reflect.Field;

public class Main {
    public static void main(String[] args) throws Exception {
        UsuarioDTO dto = new UsuarioDTO();
        dto.nome = "Alex";

        Usuario usuario = new Usuario();

        for (Field f : dto.getClass().getDeclaredFields()) {
            Field target =
usuario.getClass().getDeclaredField(f.getName());
            target.set(usuario, f.get(dto));
        }

        System.out.println(usuario.nome);
    }
}
```

🔹 Exemplo 10 – Descobrir se classe implementa interface (Plugin)

📌 Uso real

Plugins, SPI.

Interface

```
public interface Plugin {
    void executar();
}
```

Classe

```
public class PluginEmail implements Plugin {
    public void executar() {
        System.out.println("Plugin Email rodando");
    }
}
```

Main

```
public class Main {
    public static void main(String[] args) {
        Class<?> clazz = PluginEmail.class;

        for (Class<?> i : clazz.getInterfaces()) {
            if (i == Plugin.class) {
                System.out.println("É um plugin válido");
            }
        }
    }
}
```