

TÓPICO 06 - TÓPICOS ESPECIAIS EM JAVA

Disciplina de Backend - Professor Ramon Venson - SATC 2026.2

Classe `Object`

- A classe `Object` é a classe base para todas as classes em Java
- Ela define métodos que são comuns a todas as classes
- Por exemplo, o método `toString()` retorna uma representação em string de um objeto
- O método `equals()` compara dois objetos para verificar se eles são iguais

Com a classe `Object`, podemos por exemplo criar vetores que armazenam objetos genéricos:

```
// Cria um vetor de objetos genéricos
Object[] vetor = new Object[3];
// Adiciona elementos ao vetor
vetor[0] = "Hello";
vetor[1] = 123;
vetor[2] = new Object();
// Imprime o vetor
System.out.println(Arrays.toString(vetor));
// [Hello, 123, java.lang.Object@123456]
```

Construtores

- Construtores são métodos especiais que são chamados quando um objeto é criado.
- Eles são usados para inicializar os atributos do objeto.
- Os construtores têm o mesmo nome da classe e não possuem tipo de retorno.
- Se não for definido um construtor, o Java fornece um construtor padrão que não recebe parâmetros e não faz nada.
- Se você definir um construtor, o Java não fornecerá um construtor padrão.

Exemplos de Construtores

Não existe nenhuma obrigação de definir o número de parâmetros de um construtor ou sua quantidade, nem tão pouco que eles recebam os mesmos atributos da classe como parâmetro.

No entanto, em classes do tipo `POJO` (Plain Old Java Object), é comum que os construtores recebam os atributos da classe como parâmetro e façam sua inicialização.

NoArgsConstructor

O `NoArgsConstructor` é o construtor padrão que é implementado automaticamente se nenhum outro construtor for definido na classe.

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    // NoArgsConstructor - Construtor padrão  
    public Pessoa() {  
    }  
}
```

Esse construtor permite inicializar um objeto sem passar nenhum parâmetro.

AllArgsConstructor

O `AllArgsConstructor` é um construtor que recebe todos os parâmetros da classe.

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    // AllArgsConstructor - Construtor com todos os atributos  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```

Esse construtor permite inicializar um objeto com todos os atributos da classe.

Collections

`Collection` (Coleção) é uma interface que define a implementação de classes que manipulam coleções de objetos.

Essas classes são geralmente divididas em 3 tipos:

- `List` : Uma coleção ordenada que permite elementos duplicados.
- `Set` : Uma coleção que não permite elementos duplicados.
- `Queue` : Uma coleção do tipo fila (FIFO - primeiro a entrar, primeiro a sair).

Além disso, também existem as implementações da interface `Map`, que **não** são consideradas `Collections`, mas armazenam pares de chave e valor.

Implementações de `Collection`

- `ArrayList` : Uma implementação de `List` que permite acesso rápido a elementos por índice.
- `HashSet` : Uma implementação de `Set` que não permite elementos duplicados.
- `HashMap` : Uma implementação de `Map` que associa chaves a valores.
- `LinkedList` : Uma implementação de `List` que é eficiente na inserção e remoção de elementos.
- `PriorityQueue` : Uma implementação de `Queue` que ordena os elementos por prioridade.

Exceções

Exceções são eventos que ocorrem durante a execução do programa que alteram seu fluxo normal. No Java, podemos organizar exceções como:

- `Checked` : Exceções que são checadas em tempo de compilação e que exigem o tratamento ou uso da clausula `throws` . (ex.: `IOException`)
- `Unchecked` : Exceções que são geradas em tempo de execução (Ex.: `ArrayIndexOutOfBoundsException`)

Gerando exceções

Para gerar uma nova exceção no código, também podemos utilizar a cláusula throws, porém no local onde a `exception` acontece no código.

```
public int dividir(int dividendo, int divisor) {  
    if (divisor == 0) {  
        throw new ArithmeticException("Impossível dividir por zero!");  
    }  
    return dividendo / divisor;  
}
```

Tratando exceções

Há duas formas básicas de tratar Exceções. A primeira é explicitamente ignorando seu tratamento no escopo:

```
/* Usando a clausula throws neste método, garantimos
   que não será necessário realizar nenhum tipo de
   tratamento aqui, porém o erro será disparado para
   qualquer local onde a funcao leArquivo seja invocada
*/
public String leArquivo(String caminho) throws FileNotFoundException {
    Scanner scanner = new Scanner(new File(caminho));
    return scanner.nextLine();
}
```

Exceções podem ser especificadas. Ao invés de utilizar a classe `Exception`, que reconhece qualquer tipo de exceção, podemos definir exatamente que tipo de exceção pretendemos tratar:

```
public String leArquivo() throws FileNotFoundException {  
    Scanner scanner = new Scanner(new File("example.txt"));  
    return scanner.nextLine();  
}
```

`FileNotFoundException` é gerada quando o arquivo não está presente no sistema de arquivos

Para tratar uma exceção e evitar que ela seja transmitida para cima. Podemos utilizar o bloco `try-catch` :

```
public void leArquivo() {  
    try {  
        Scanner scanner = new Scanner(new File("example.txt"));  
        System.out.println(scanner.nextLine());  
    } catch (FileNotFoundException exception) {  
        System.out.println(exception);  
    }  
}
```

Dessa forma, sempre que a exceção ocorrer, o código dentro da clausula `catch` será executado e o programa não será interrompido.

Terminologias

- `Tempo de execução` - aquilo que acontece enquanto o programa roda
- `Tempo de compilação` - o que acontece quando o programa é compilado
- `Checked Exception` - exceções que são verificadas em tempo de compilação
- `Unchecked Exception` - exceções que são verificadas em tempo de execução
- `Runtime Exception` - exceções que são verificadas em tempo de execução e que não precisam ser tratadas

Outros Pontos para Estudar

- **Generics:** permite criar estruturas que funcionam com qualquer tipo de dado
- **Streams:** permite processar sequências de elementos de forma funcional
- **Threads:** permite a execução de múltiplas tarefas simultaneamente
- **Logs:** permite registrar eventos ocorridos durante a execução do programa
- **Files e HTTP Requests:** permitem manipular arquivos e realizar requisições HTTP
- **Build Tools (Maven, Gradle):** permitem automatizar o processo de build e gerenciamento de dependências