AULA 06 - TÓPICOS ESPECIAIS EM JAVA

Disciplina de Backend - Professor Ramon Venson - SATC 2024

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.

NoArgsConstrutor

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;
    // NoArgsConstrutor - Construtor padrão
    public Pessoa() {
    }
}
```

Esse construtor permite inicializar um objeto sem passar nenhum parâmetro.

AllArgsConstrutor

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.

Destrutores

- Destrutores são métodos especiais que são chamados automaticamente quando um objeto é destruído.
- Eles são usados para liberar recursos alocados pelo objeto.
- Os destrutores não possuem tipo de retorno.
- Os destrutores não são herdados.
- Os destrutores não podem ser sobrecarregados.
- Os destrutores não podem ser chamados explicitamente.

Exemplo de Destrutor

Um destrutor pode ser implementado usando o mesmo nome da classe, precedido pelo símbolo `~`:

```
public class Pessoa {
    // Destrutor
    ~Pessoa() {
        RegistroGeral.deletar(this);
    }
}
```

Esse método será chamado automaticamente quando o objeto em questão for destruído.

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 que associa chaves a valores.

Além disso, também existem as implementações da interface `Map` (árvores), que não são consideradas `Collections` mas podem ser manipuladas como tal.

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 ass
- `LinkedList`: Uma implementação de `List` que permite acesso rápido a elementos por índice.
- `PriorityQueue`: Uma implementação de `Queue` que permite acesso rápido a elementos por índice.

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.: `ArrayIndexOutOfBounds`)

Gerando exceções

Para gerar uma nova excessão no código, também podemos utilizar a clausula throws, porem 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 void leArquivo(String caminho) throws Exception{
    File file = new File("example.txt");
    return file;
}
```

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 void leArquivo() throws FileNotFoundException{
    File file = new File("example.txt");
    return file;
}
```

> `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 {
        File file = new File("example.txt");
    } catch (FileNotFoundException exception) {
        System.out.println(exception);
    }
    return file;
}
```

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

Outros Pontos

- Generics
- Streams
- Collections
- Threads
- Logs
- Data Structures
- Files e HTTP Requests
- Build Tools