

Threads

Definição

- ✓ Unidade básica de execução
- ✓ Concorrência: várias tarefas ao mesmo tempo
 - ✓ Processamento paralelo
- ✓ Multithreading: várias threads em um processo

Tradução Literal

Thread equivale a uma "linha" de execução.

Aplicação

- ✓ Processamento Paralelo: aplicações de servidores com múltiplos clientes, jogos, etc.
- ✓ Operações Assíncronas: tarefas em segundo plano, evita bloqueio da função principal.
- ✓ UI Responsiva: manter interface interativa durante o processamento.

Características

- ✓ Concorrência: várias threads ocupam a mesma memória
- ✓ Desempenho Paralelo: execução de tarefas em paralelo
- ✓ Controlado por start(), sleep(), join(), interrupt()
- ✓ Ciclo de Vida: Novo, Executável, Executando, Bloqueado, e Morto.

Vantagens

- ✓ Melhor Utilização de Recursos do Sistema
- ✓ Responsividade
- ✓ Execução Concorrente

Desvantagens

- ✓ Sincronização Complexa
- ✓ Overhead de Mudança de Contexto
- ✓ Dificuldade de Debugging

Implementação

Exemplo

✓ Estendendo a classe Thread

```
class MinhaThread extends Thread {  
    public void run() {  
        System.out.println("Thread em execução");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        MinhaThread t1 = new MinhaThread();  
        t1.start();  
    }  
}
```

✓ Implementando a interface Runnable

```
class MinhaRunnable implements Runnable {  
    public void run() {  
        System.out.println("Thread em execução");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Thread t = new Thread(new MinhaRunnable());  
        t.start();  
    }  
}
```

Função Lambda

Definição

- ✓ Java 8
- ✓ Forma de definir função
- ✓ Permite passagem de blocos de código como parâmetro
- ✓ Compacta funções
- ✓ Útil para funções pequenas e temporárias

Tradução Literal

Lambda em programação é o mesmo que função anônima

Características

- ✓ Anônima : não tem nome
- ✓ Compacta : reduz quantidade de código
- ✓ Usado em interfaces funcionais : classes com métodos abstratos como ActionListener
- ✓ Inferência : o compilador deduz o tipo de parâmetro
- ✓ Acessam Variáveis do Escopo Externo - closure

Sintaxe

(parâmetros) -> expressão;
(parâmetros) -> { expressões; }

Vantagens

- ✓ Código Mais Limpo e Conciso
- ✓ Eliminação de Classes Internas Anônimas
- ✓ Uso Eficiente com Streams
- ✓ Redução de Overhead
- ✓ Melhor Legibilidade

Desvantagens

- ✓ Dificuldade Legibilidade em Casos Complexos
- ✓ Dificuldade de Debugging
- ✓ Limitações de Expressividade: geralmente limitado a uma expressão ou bloco de expressões

Aplicação

- ✓ Ordenação de Coleções: Usadas com Comparator para ordenar coleções.
- ✓ Iteração sobre Coleções: Usadas com forEach para iterar sobre listas e streams.
- ✓ Manipulação de Streams: úteis com a API em operações como filter, map, reduce.
- ✓ Listeners em GUI: simplificar a criação de listeners

Exemplo

```
button.addActionListener(e -> System.out.println("Botão clicado!"));

List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> pares = numeros.stream().filter(n -> n % 2 == 0).collect(Collectors.toList());
System.out.println(pares);

List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);
numeros.forEach(n -> System.out.println(n));
```