📘 Aula – Threads e Concorrência em Java (30 minutos)

🎯 Objetivo da aula

Ao final da aula, o aluno deve:

- Entender **o que é Thread**
- Saber **quando e por que usar concorrência**
- Conhecer **vantagens e riscos**
- Aplicar **Threads em problemas reais**
- Diferenciar **execução sequencial x concorrente**

⏱️ Sugestão de tempo

- **10 min** → Teoria e conceitos
- **5 min** → Vantagens, cuidados e quando usar
- **15 min** → Exemplos práticos (5 exemplos)

🧠 PARTE 1 – TEORIA (10 min)

O que é uma Thread?

Thread é uma **linha de execução dentro de um programa**.

👉 Um programa Java **sempre começa com uma thread**, chamada **Thread principal (main)**.

Sem threads:

- Uma tarefa termina → depois a outra começa

Com threads:

- Várias tarefas **executam ao mesmo tempo**

Processo x Thread (explicação simples)

- **Processo** → Programa rodando (JVM)
- **Thread** → Tarefas dentro do programa

🧠 Analogia:

- Processo = Restaurante
- Threads = Garçons atendendo mesas ao mesmo tempo

Concorrência é a capacidade de:

> Executar **mais de uma tarefa ao mesmo tempo** ou de forma intercalada

Em Java isso é feito com:

- `Thread`
- `Runnable`
- `ExecutorService`

Quando usar Threads?

Use quando:

- Tarefas são **independentes**
- Há **espera** (rede, arquivo, banco)
- Você quer **ganhar desempenho**

- Evitar travar a aplicação

Não use quando:

- A lógica é simples
- Não há ganho real de desempenho
- O código pode ficar mais complexo sem necessidade

⚠️ PARTE 2 – VANTAGENS E CUIDADOS (5 min)

✅ Vantagens

- Melhor desempenho
- Aproveita múltiplos núcleos do CPU
- Interface mais responsiva
- Processamento paralelo

❌ Cuidados

- Concorrência mal feita gera:
  - Erros difíceis de achar
  - Dados inconsistentes
  - Deadlock
- Mais código ≠ melhor código

👉 **Regra de ouro:**

> Use threads quando há ganho real

🧪 PARTE 3 – EXEMPLOS PRÁTICOS (15 min)

Todos os exemplos abaixo:
✔ Mundo real
✔ Classe `main`
✔ Código simples e explicação clara

🧩 EXEMPLO 1 – Caixa de supermercado (duas filas)

Problema real

Dois caixas atendendo clientes ao mesmo tempo.

Código

```
java
public class CaixaSupermercado {

    public static void main(String[] args) {

        Runnable caixa1 = () -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Caixa 1 atendendo cliente " + i);
                dormir();
            }
        };

        Runnable caixa2 = () -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Caixa 2 atendendo cliente " + i);
                dormir();
            }
        };

        new Thread(caixa1).start();
        new Thread(caixa2).start();
    }

    private static void dormir() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Explicação

- Dois caixas = duas threads

- Ambos atendem ao mesmo tempo
- `sleep` simula tempo de atendimento

🧩 EXEMPLO 2 – Download de arquivos

Problema real

Baixar vários arquivos ao mesmo tempo.

Código

```
java
public class DownloadArquivos {

    public static void main(String[] args) {

        Runnable download = () -> {
            String nome = Thread.currentThread().getName();
            System.out.println("Iniciando download: " + nome);
            dormir();
            System.out.println("Finalizando download: " + nome);
        };

        new Thread(download, "Arquivo-1.zip").start();
        new Thread(download, "Arquivo-2.zip").start();
        new Thread(download, "Arquivo-3.zip").start();
    }

    private static void dormir() {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Explicação

- Cada arquivo é um download independente
- Downloads acontecem simultaneamente

- Muito comum em sistemas reais

🧩 EXEMPLO 3 – Envio de e-mails em paralelo

Problema real

Enviar vários e-mails sem travar o sistema.

Código

```
java
public class EnvioEmail {

    public static void main(String[] args) {

        for (int i = 1; i <= 3; i++) {
            int email = i;

            new Thread(() -> {
                System.out.println("Enviando e-mail " + email);
                dormir();
                System.out.println("E-mail " + email + " enviado");
            }).start();
        }
    }

    private static void dormir() {
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Explicação

- Cada e-mail roda em uma thread
- Sistema continua funcionando

- Muito usado em backend

🧩 EXEMPLO 4 – Atendimento de clientes (chat)

Problema real

Cada cliente atendido por um atendente.

Código

```
java
public class AtendimentoChat {

    public static void main(String[] args) {

        Runnable cliente1 = () -> atendimento("Cliente 1");
        Runnable cliente2 = () -> atendimento("Cliente 2");

        new Thread(cliente1).start();
        new Thread(cliente2).start();
    }

    private static void atendimento(String cliente) {
        System.out.println("Atendendo " + cliente);
        dormir();
        System.out.println("Finalizado atendimento de " + cliente);
    }

    private static void dormir() {
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Explicação

- Cada cliente é uma thread
- Atendimento simultâneo

- Base de sistemas de chat e suporte

🧩 EXEMPLO 5 – Processamento de pedidos (e-commerce)

Problema real

Processar vários pedidos ao mesmo tempo.

Código

```
java
public class ProcessamentoPedidos {

    public static void main(String[] args) {

        for (int pedido = 1; pedido <= 4; pedido++) {
            int numeroPedido = pedido;

            new Thread(() -> {
                System.out.println("Processando pedido " +
numeroPedido);
                dormir();
                System.out.println("Pedido " + numeroPedido + "
finalizado");
            }).start();
        }
    }

    private static void dormir() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Explicação

- Cada pedido roda em paralelo
- Sistema escala melhor

- Muito comum em e-commerce e APIs

🧠 FECHAMENTO DA AULA (2 min)

Mensagem final para os alunos

> Thread não é para complicar
> Thread é para **resolver problemas reais de desempenho**

Próximo passo sugerido

- `ExecutorService`
- Pool de threads
- Sincronização (`synchronized`, `Lock`)