Aqui está uma **aula completa de 1 hora sobre *break* e *continue* em Java**, estruturada para ensino profissional, com teoria, exemplos, vantagens, problemas e boas práticas.

AULA COMPLETA (1 hora) — Break e Continue em Java

🎯 Objetivo da aula

Ensinar de forma profissional os comandos de controle de fluxo **break** e **continue** em Java, aplicando em cenários práticos, loops e estruturas reais de mercado.

---

1. Introdução (5 minutos)

✅ O que são comandos de controle de fluxo?

São instruções especiais que modificam o fluxo padrão de execução do código, como:

- Parar um laço antes do tempo.
- Pular uma iteração.
- Encerrar estruturas.

No Java, os dois comandos principais para controle em loops são:

- **break**
- **continue**

---

2. O comando BREAK (15 minutos)

📌 Conceito

O comando **break** encerra imediatamente:

- O loop atual (`for` , `while` , `do-while`)
- Ou um bloco `switch`

Após o break, o código continua na próxima linha **fora** do bloco encerrado.

📌 Quando usar?

- Finalizar a busca quando o resultado for encontrado.
- Evitar processamento desnecessário.
- Sair de laços com erro ou condição inesperada.

⚠️ Problemas comuns com break

1. **Quebrar loops cedo demais**, gerando resultados incompletos.
2. Uso excessivo, deixando o fluxo confuso.
3. Dificuldade de manutenção caso existam muitos break espalhados.

⭐ Vantagens

- Otimiza desempenho evitando execuções desnecessárias.
- Clareza em loops simples.
- Útil para buscas e validações.

---

3. O comando CONTINUE (15 minutos)

📌 Conceito

O comando **continue** pula a iteração atual e volta para o início do loop.

Não encerra o loop — apenas **ignora** o restante do bloco da iteração atual.

📌 Quando usar?

- Ignorar valores inválidos.
- Pular elementos específicos.
- Tratar listas onde apenas alguns itens importam.

⚠️ Problemas comuns com continue

1. Ignorar lógica sem perceber.

2. Deixar o fluxo difícil de seguir.
3. Criar loops infinitos se não atualizar variáveis corretamente.

⭐ Vantagens

- Código mais limpo quando queremos pular casos pontuais.
- Evita aninhamento profundo de IFs.
- Facilita filtros e validações durante loops.

---

4. 8 Exemplos Práticos (25 minutos)

✅ EXEMPLO 1 – Uso básico do BREAK em um loop FOR

Problema

Encontrar o primeiro número maior que 50 em um array.

Código

java Copiar código

```java
int[] numeros = {10, 20, 30, 40, 55, 60, 70};

for (int n : numeros) {
    if (n > 50) {
        System.out.println("Primeiro maior que 50: " + n);
        break; // Encerra o loop aqui
    }
}
```

Explicação

Assim que o 55 é encontrado, não faz sentido continuar buscando.

Vantagem

Melhor performance ao evitar iterações desnecessárias.

✅ EXEMPLO 2 – BREAK em busca de usuário

Problema

Parar ao encontrar um nome específico.

Código

java Copiar código

```java
String[] nomes = {"Ana", "Lucas", "Marcos", "João"};

for (String nome : nomes) {
    if (nome.equals("Marcos")) {
        System.out.println("Usuário encontrado!");
        break;
    }
}
```

Vantagem

Busca eficiente em lista.

✅ EXEMPLO 3 – BREAK para interromper leitura em menu

Problema

Usuário escolhe opção 0 para sair.

Código

java Copiar código

```java
Scanner sc = new Scanner(System.in);

while (true) {
    System.out.println("1 - Comprar");
    System.out.println("2 - Vender");
    System.out.println("0 - Sair");
    int opcao = sc.nextInt();

    if (opcao == 0) {
```

```
            System.out.println("Encerrando o sistema...");
            break;
        }

        System.out.println("Processando opção " + opcao);
    }
```

Vantagem

Simples e direto para loops infinitos controlados por break.

✅ EXEMPLO 4 – CONTINUE para ignorar números negativos

Problema

Calcular a soma apenas dos números positivos.

Código

java Copiar código

```java
int[] valores = {10, -5, 20, -3, 7};
int soma = 0;

for (int v : valores) {
    if (v < 0) {
        continue; // pula números negativos
    }
    soma += v;
}

System.out.println("Soma dos positivos: " + soma);
```

Vantagem

Evita IF aninhado e deixa o loop limpo.

✅ EXEMPLO 5 – CONTINUE para filtrar erros em uma lista

Problema

Processar apenas emails válidos.

Código

java Copiar código

```java
String[] emails = {"alex@", "teste@gmail.com", "invalido", "user@hotmail.com"};

for (String email : emails) {
    if (!email.contains("@")) {
        continue; // ignora e-mail inválido
    }
    System.out.println("Enviando email para: " + email);
}
```

✅ EXEMPLO 6 – BREAK com loops aninhados (break rotulado)

Problema

Parar completamente quando achar um número.

Código

java Copiar código

```java
int[][] matriz = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

procura:
for (int[] linha : matriz) {
    for (int num : linha) {
        if (num == 5) {
            System.out.println("Número encontrado!");
            break procura; // sai de AMBOS os loops
        }
```

```
        }
    }
```

Explicação

Sem rótulo, o break sairia apenas do loop interno.

✅ EXEMPLO 7 – CONTINUE com FOR para tratar pares/ímpares

Problema

Mostrar apenas números pares.

Código

java Copiar código

```java
for (int i = 1; i <= 10; i++) {
    if (i % 2 != 0) {
        continue; // ignora ímpares
    }
    System.out.println("Par: " + i);
}
```

✅ EXEMPLO 8 – BREAK e CONTINUE juntos em validações reais

Problema

Processar pedidos:

- ignorar pedidos com valor negativo (continue)
- parar se encontrar pedido acima de 10 mil (break)

Código

java Copiar código

```java
int[] pedidos = {100, 200, -50, 300, 15000, 400};

for (int p : pedidos) {
```

```
        if (p < 0) {
            System.out.println("Pedido inválido ignorado: " + p);
            continue;
        }

        if (p > 10000) {
            System.out.println("Pedido alto demais! Encerrando.");
            break;
        }

        System.out.println("Processando pedido: " + p);
    }
```

---

5. Boas práticas (5 minutos)

✔ Quando usar BREAK

- Interromper buscas.
- Sair de loops infinitos.
- Parar execução ao encontrar erro.

✔ Quando usar CONTINUE

- Ignorar itens não desejados.
- Filtrar durante loops.
- Evitar IFs aninhados.

✘ Quando NÃO usar

- Quando confunde o fluxo.
- Quando pode ser substituído por lógica clara.
- Quando cria código difícil de manter.