1. Classe `Deposito`

Esta classe representa o armazenamento onde o produtor coloca e o consumidor retira caixas. Ela possui:

- **Atributos**:
  - `items`: variável que mantém o número de caixas no depósito (inicialmente 0).
  - `capacidade`: capacidade máxima do depósito, definida como 10 caixas.
- **Métodos**:
  - `retirar()`: diminui o número de caixas do depósito em 1, se ainda houver caixas disponíveis. Retorna 1 se a retirada foi bem-sucedida, ou 0 se o depósito estava vazio.
  - `colocar()`: aumenta o número de caixas no depósito em 1, se ele ainda não atingiu a capacidade máxima. Retorna 1 se a caixa foi armazenada com sucesso, ou 0 se o depósito já estava cheio.

---

2. Classe `Produtor`

Esta classe representa uma **thread** que produz caixas e as coloca no depósito em intervalos regulares de tempo.

- **Atributos**:
  - `deposito`: referência ao objeto `Deposito` compartilhado entre o produtor e o consumidor.
  - `tempoEntreProducao`: intervalo (em segundos) entre cada produção de caixa.
- **Construtor**:
  - Recebe o objeto `Deposito` e o tempo entre produções como parâmetros.
- **Método `run()`**:
  - O método `run()` é executado automaticamente quando a thread é iniciada e contém a lógica de produção.
  - O loop `while (true)` permite que o produtor produza continuamente, a cada `tempoEntreProducao` segundos.
  - A cada iteração do loop:
    - Chama `deposito.colocar()`, que tenta adicionar uma caixa ao depósito.
    - Usa `Thread.sleep(tempoEntreProducao * 1000);` para pausar a execução por um período especificado.

---

3. Classe `Consumidor`

Esta classe representa uma **thread** que consome (retira) caixas do depósito em intervalos regulares de tempo. Ela herda diretamente de `Thread`.

- **Atributos**:
  - `deposito`: referência ao objeto `Deposito`.
  - `tempoEntreConsumo`: intervalo (em segundos) entre cada consumo de caixa.
- **Construtor**:
  - Recebe o objeto `Deposito` e o tempo entre consumos como parâmetros.
- **Método `run()`**:
  - O método `run()` contém a lógica de consumo, executada quando a thread é iniciada.
  - Assim como em `Produtor`, o loop `while (true)` permite que o consumidor consuma continuamente.
  - A cada iteração:
    - Chama `deposito.retirar()`, que tenta retirar uma caixa do depósito.
    - Usa `Thread.sleep(tempoEntreConsumo * 1000);` para aguardar o tempo configurado.

---

4. Classe `Main`

A classe principal do programa (`Main`) serve para iniciar o processo de produção e consumo.

- **Método `main()`**:
  - Cria uma instância do `Deposito`, compartilhada entre o `Produtor` e o `Consumidor`.
  - Cria um objeto `Produtor` e o configura para adicionar uma caixa ao depósito a cada 2 segundos.
  - Cria um objeto `Consumidor` e o configura para retirar uma caixa do depósito a cada 1 segundo.
  - Inicia as threads para `Produtor` e `Consumidor`:
    - Usa `new Thread(p).start();` para iniciar o `Produtor`, pois ele implementa `Runnable`.
    - Usa `c.start();` para iniciar o `Consumidor`, pois ele herda de `Thread` e pode ser iniciado diretamente.

---

5. Fluxo do Programa

- A `Main` inicializa as duas threads:
  - O **Produtor** adiciona uma caixa ao `Deposito` a cada 2 segundos, desde que haja espaço.
  - O **Consumidor** retira uma caixa do `Deposito` a cada 1 segundo, desde que haja caixas disponíveis.
- Assim, o **Produtor e o Consumidor** operam simultaneamente no depósito, de forma a simular uma relação de produção e consumo.