1. **Tres esteiras com grupos de funcionários**:
   * **Esteira 1**: Possui 4 grupos de 12 funcionários, cada grupo produzindo uma parte específica do robô (braço esquerdo, braço direito, perna direita, perna esquerda). Implementamos GrupoEsteira1 com lógica para 4 grupos.
   * **Esteira 2**: Possui 2 grupos de 9 funcionários, um produzindo a carcaça e o outro a cabeça do robô. Implementamos GrupoEsteira2 com lógica para 2 grupos.
   * **Esteira 3**: Possui 3 grupos de 5 funcionários, responsáveis pela montagem do robô usando peças das esteiras 1 e 2. Implementamos GrupoEsteira3 com lógica para montagem usando ferramentas.
2. **Funcionamento circular**:
   * Utilizamos ArrayBlockingQueue para implementar buffers circulares nas esteiras 1 e 2, garantindo que as partes são produzidas e consumidas de maneira circular.
3. **Produção e montagem autônoma**:
   * Cada funcionário nos grupos das esteiras 1 e 2 é capaz de produzir partes do robô de forma autônoma. Isso é gerenciado através de contadores circulares (AtomicInteger).
4. **Utilização de ferramentas e estratégia para evitar deadlock**:
   * Na esteira 3, os funcionários utilizam ferramentas adjacentes para montar o robô. Implementamos a lógica para bloquear e desbloquear as ferramentas adjacentes usando Lock para evitar deadlock.
5. **Não usar ferramentas para coleta de peças**:
   * A coleta de peças das esteiras 1 e 2 é feita sem o uso das ferramentas. As ferramentas são usadas apenas para a montagem do robô.
6. **Paralelismo na coleta de peças**:
   * Utilizamos threads e buffers circulares para simular o paralelismo na coleta de peças, garantindo que as partes são buscadas de maneira eficiente.
7. **Logs detalhados**:
   * O objeto Robo contém informações detalhadas sobre a posição das partes nas esteiras, os grupos e funcionários que as produziram, e detalhes sobre a montagem. Os logs incluem todas essas informações, garantindo rastreabilidade completa.

### **Justificação**

* **Estrutura de funcionários e produção**: A implementação de GrupoEsteira1, GrupoEsteira2 e GrupoEsteira3 garante que a produção é feita por grupos de funcionários com contagem circular, conforme descrito.
* **Circularidade das esteiras**: A utilização de ArrayBlockingQueue implementa buffers circulares, permitindo que a produção e consumo de partes sejam contínuos e circulares.
* **Autonomia na produção**: Cada grupo de funcionários trabalha de forma independente, produzindo partes específicas sem depender de uma ordem sequencial.
* **Evitar deadlocks**: A lógica de bloqueio e desbloqueio de ferramentas na esteira 3 previne deadlocks, permitindo que a montagem ocorra sem interrupções.
* **Paralelismo e coleta de peças**: A coleta de peças ocorre em paralelo usando threads, com as peças sendo colocadas nas esteiras de forma eficiente.

### **—----------------------------------------------------------------------------------------------------- Logger**

**Função:** log(String message)

* **Propósito:** Registrar eventos importantes no sistema, como a produção de partes e a montagem de robôs.
* **Justificação:** É crucial ter um registro detalhado para fins de monitoramento, debug e auditoria. O método é sincronizado para garantir a integridade dos logs quando acessado por múltiplas threads simultaneamente. A remoção de acentuação garante a compatibilidade e legibilidade dos logs.

### **ParteRobo**

**Funções:** Construtor e métodos getters (getTipo, getPosicaoEsteira, getGrupo, getFuncionario)

* **Propósito:** Representar as partes individuais do robô, armazenando informações sobre seu tipo, posição na esteira, grupo e funcionário responsável pela produção.
* **Justificação:** Permite encapsular as informações relevantes de cada parte do robô, facilitando o rastreamento e a montagem final, além de proporcionar um log detalhado.

### **GrupoEsteira1, GrupoEsteira2, GrupoEsteira3**

**Função:** run()

* **Propósito:** Implementar a lógica de produção de partes e montagem dos robôs, sendo executada por threads.
* **Justificação:** O método run é essencial para a execução concorrente das tarefas de produção e montagem. Utilizamos threads para simular o trabalho paralelo dos funcionários. O AtomicInteger é usado para garantir que os funcionários sejam contados de maneira circular e thread-safe. O ArrayBlockingQueue serve como buffer circular, garantindo a produção e consumo contínuos e sincronizados.

### **GrupoEsteira3**

**Função:** run()

* **Propósito:** Realizar a montagem final dos robôs utilizando as partes produzidas nas esteiras 1 e 2.
* **Justificação:** A lógica de montagem inclui bloqueio e desbloqueio de ferramentas (Lock), essencial para evitar deadlocks quando múltiplos funcionários tentam acessar as mesmas ferramentas simultaneamente. A montagem ocorre apenas quando as peças necessárias estão disponíveis, e as ferramentas são usadas apenas para a montagem, não para a coleta das peças.

### **FabricaRoboMain**

**Função:** main(String[] args)

* **Propósito:** Inicializar o sistema de produção, criando e iniciando threads para os grupos de funcionários em cada esteira.
* **Justificação:** Serve como ponto de entrada para a aplicação, configurando os buffers circulares (ArrayBlockingQueue) e as ferramentas (Lock), e iniciando os threads que representam os grupos de produção e montagem. Garante que todos os componentes do sistema sejam corretamente configurados e que a produção inicie de forma coordenada.

### **Robo**

**Funções:** Construtor e toString()

* **Propósito:** Representar o robô montado com todas as suas partes, armazenando informações detalhadas sobre a origem de cada peça e o processo de montagem.
* **Justificação:** O construtor recebe todas as partes necessárias e as informações dos funcionários que realizaram a montagem, encapsulando essas informações em um único objeto. O método toString fornece uma descrição detalhada do robô montado, usada nos logs para rastrear todo o processo de produção e montagem.

### **Utilização de Locks**

**Ferramentas para montagem na esteira 3**

* **Propósito:** Garantir que dois funcionários não utilizem as mesmas ferramentas ao mesmo tempo durante a montagem dos robôs.
* **Justificação:** Evitar deadlocks e condições de corrida, assegurando que a montagem dos robôs seja realizada de forma segura e ordenada. O uso de Lock permite controle fino sobre a sincronização das ferramentas, desbloqueando-as assim que a montagem é concluída.

Cada uma dessas funções foi projetada para atender a requisitos específicos da simulação de uma fábrica de robôs, garantindo produção eficiente, montagem sem conflitos e rastreamento detalhado dos processos. O uso de estruturas de dados concorrentes, como ArrayBlockingQueue e AtomicInteger, juntamente com técnicas de sincronização, como Lock, foi fundamental para garantir que o sistema funcione de maneira segura e eficiente em um ambiente multi-thread.