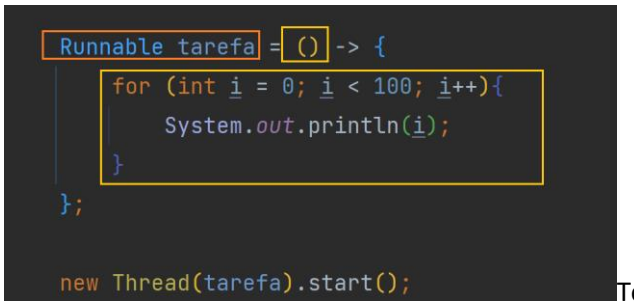
Interface funcional

Beleza, **um lambda pode ser compilado e entendido** porque a **interface** que usamos por baixo do pano possuí apenas um método *(então o compilador já sabe onde encaixar o código do lambda).*

Isso funciona até mesmo pra interfaces pré-java 8, tipo a **Runnable**(Interface que executa algo em thread paralela) que pode ter **seu comportamento escrito como uma lambda**:



Todas as interfaces que possuem um único método se enquadram como **interfaces funcionais** *(Comparable, Runnable etc são interfaces funcionais, mesmo antes desse conceito existir, pois elas já eram interfaces com um único método)*.

@FunctionalInterface

Se você não quiser usar a maneira implícita de declarar uma interface funcional (ter um único método abstrato) você pode usar a anotação **@FunctionalInterface**, que deixa explicito esse comportamento de uma interface *(e claro, serve como uma validadora caso alguém tente adicionar outro método)*:

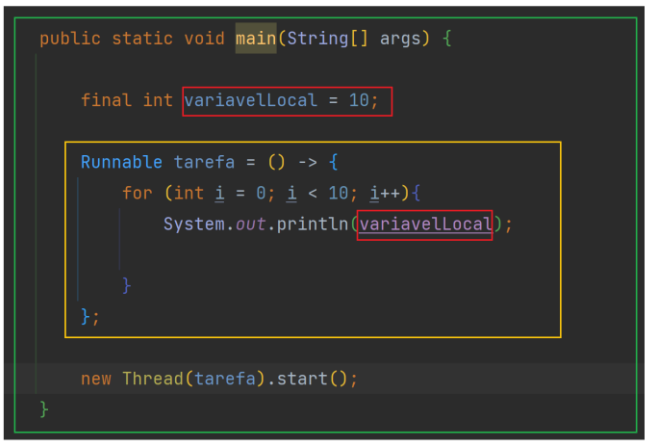
```
@FunctionalInterface
public interface Validador<T> {

    boolean valida(T t);
}
```

Lambdas não existem se não forem atribuídas/inferidas em **Interfaces funcionais,** a sintaxe da lambda sabe quando ela pode ou não ser inferida, pois ela já espera o tipo compatível.

Variáveis na lambda

É possível acessar as **variáveis finais** *(você pode não as declarar como finais, mas não pode mudar elas depois de usar numa lambda)* **locais no método** onde a **lambda** está **presente**:


public static void main(String[] args) {
final int variavelLocal = 10;
Runnable tarefa = () -> {
for (int i = 0; i < 10; i++){
System.out.println(variavelLocal);
}
};
new Thread(tarefa).start();
}