A expressão lambda é uma função sem declaração já que nela não é necessário colocar um nome, um tipo de retorno e um modificador de acesso; o método é declarado no mesmo lugar em que será usado. A vantagem de funções lambda é diminuir a quantidade de código necessária para a escrita de algumas funções como algumas classes internas (Listeners e Threads).

A função lambda pode ter nenhum ou vários parâmetros e seus tipos podem ser colocados ou podem ser omitidos; pode também ter nenhum ou vários comandos: com um comando, chaves não são obrigatórias e a função retorna o valor calculado na expressão; se tiver vários comandos, deve-se colocar chaves e o comando return (caso nada seja retornado, a função tem um retorno void).

Os principais tipos de utilização da função lambda são: função lambda com Threads, função lambda com as classes de Collections, função lambda utilizada com Listeners e, por fim, métodos que aceitam funções lambdas como parâmetros.

Um exemplo de uma função lambda com Threads é criar uma thread com uma função interna. Para isso, deve-se seguir os passos: criar uma implementação do método run da interface runnable e em seguida, criar a thread com essa implementação; com a utilização de expressões lambda, pode-se perceber que essa função pode ser definida e passada como parâmetros diretamente para outros métodos. Também pode-se utilizar funções lambda com classes de coleções do Java (Collections): serão feitos variados tipos de funções que percorrerão a coleção e farão determinada ação como imprimir todos os elementos da coleção, filtrar elementos da lista ou buscar um determinado valor na lista. Listeners são classes que implementam o padrão de projeto Observer, que representa objetos que ficam esperando ações realizadas em outros objetos e, a partir dessa ação, executam algum código; um exemplo são os Listeners de botões da API de interfaces gráficas Swing. Para criar métodos que recebem funções lambda como parâmetro, pode-se por exemplo, criar um método genérico para imprimir elementos de uma lista e passar como parâmetro a função para a filtragem dos elementos dessa lista; assim, com apenas um método, e passando a função como parâmetro, é possível fazer a filtragem da lista de várias maneiras diferentes.

Threads são subdivisões dos processos, ou seja, cada processo possui diversas threads (linhas de instrução). Dessa maneira, pode-se dividir partes do programa em Java para trabalhar paralelamente. As threads estão em todos os processamentos realizados em um computador.

Em um programa em Java, pode-se querer executar dois ou mais threads ao mesmo tempo, ou seja, dois ou mais procedimentos internos do programa.

Um exemplo é um cálculo que faz consultas a um Web Service: ele vai receber uma lista com mais ou menos 70 mil registros, depois pega o valor de cada registro e soma. Após isso, captura um valor para reajuste em porcentagem (capturado de um WebService que pode demorar a responder), e recalcula o valor somado. Por fim retorna o resultado via WebService. Esse procedimento pode demorar minutos, e não vai retornar nada ao usuário, apenas fazer comunicação interna entre Web Services. Seria de péssima viabilidade parar toda a aplicação para executar esse procedimento ou por exemplo, para fazer um cadastro simples, ter que esperar o tempo dele fazer o processamento. Utilizando o conceito de Threads, é possível melhorar. A solução então é fazer com que esse procedimento seja executado concorrentemente, ou seja, ao mesmo tempo em que o usuário está realizando o cadastro, o procedimento também é executado, e provavelmente quando ele terminar o cadastro, o procedimento também já terminou, de forma imperceptível a ele.

Existe uma interface chamada Runnable que possui um método run. Dentro do método run devem ficar os procedimentos que serão executardos paralelamente, sendo assim vamos coloca-se todo o código dentro de um método run. Como Runnable é apenas um contrato, precisa-se de alguma classe que a implemente e faça o trabalho da “paralelização”, que é a classe Thread. Quando fizer o “.start();” já estará iniciando o processamento paralelo, e liberando o programa para executar qualquer outra thread. Logo, quando deseja-se que o programa não “trave para o usuário” naquele determinado procedimento que pode demorar muito tempo, use Thread. Caso deseja-se que uma classe seja processada paralelamente, porém ela já estende de outra, opta-se por implementar o Runnable, que é a interface padrão para Thread. Por boas práticas, geralmente implementa-se Runnable em vez de estender de Thread.