Qual o resultado do seguinte programa?

```
public class MathFunctions {
    public static void addToInt(int x, int amountToAdd) {
        x = x + amountToAdd;
    }

    public static void main(String[] args) {
        int a = 15;
        int b = 10;
        MathFunctions.addToInt(a, b);
        System.out.println(a);
    }
}
```

A. 10
B. 15
C. 25
D. Erro de compilação na linha 3
E. Erro de compilação na linha 8
F. Nenhuma das alternativas acima

EXPLICAÇÃO DO RESULTADO

O resultado do programa será **B. 15**.

Vamos analisar o programa passo a passo:

1. Na classe `MathFunctions`, há um método estático `addToInt` que recebe dois parâmetros `x` e `amountToAdd`.
2. Dentro do método `addToInt`, a linha `x = x + amountToAdd;` soma `amountToAdd` ao valor de `x`. Esta operação é feita localmente dentro do método `addToInt`.
3. No método `main`:
   - `a` é inicializado com o valor `15`.
   - `b` é inicializado com o valor `10`.
   - `MathFunctions.addToInt(a, b);` é chamado. Aqui, `a` (que vale `15`) é passado como `x` e `b` (que vale `10`) é passado como `amountToAdd`.
4. Dentro do método `addToInt`, `x` recebe `x + amountToAdd`, ou seja, `x` dentro do método se torna `25` (pois `15 + 10 = 25`).
5. No entanto, a modificação feita em `x` dentro do método `addToInt` **não afeta a variável `a` no método `main`. Isso ocorre porque em Java os argumentos são passados por valor**, ou seja, uma cópia do valor de `a` é passada para `x`.
6. Após `MathFunctions.addToInt(a, b);` ser chamado, o valor de `a` no método `main` ainda é `15`.
7. Por fim, `System.out.println(a);` imprime o valor de `a`, que continua sendo `15`.

Portanto, o resultado final do programa será **15** (alternativa B).

Entretanto em Java, quando você passa um **objeto como argumento** para um método, **você está passando a referência para esse objet**o. Neste caso isso significa que quaisquer modificações feitas no objeto dentro do método afetarão o objeto original fora do método.