

# Introdução à Linguagem Java

## Testes com JUnit

Lista 03

4 de maio de 2021

1. Crie uma classe **Cliente** com os campos nome, saldo e número da conta. Esta classe deve:
  - Gerar o número da conta dos clientes automaticamente a partir do número 1001;
  - Ter dois construtores diferentes, um que recebe apenas o nome do cliente, e um segundo que recebe o nome e o saldo. Reaproveite o código para os construtores;
  - Fornecer métodos void para saque, depósito e impressão dos dados;
  - Escreva um método que verifica se o objeto em questão tem um saldo maior ou igual a um saldo dado;
  - Faça um método que verifica se o objeto em questão tem um nome igual ao passado como parâmetro.

Teste o funcionamento desta classe usando o Junit, ao menos um teste por método.
2. Mude o nome da classe para **Cliente2**. Copie, renomeie e aumente a funcionalidade da classe anterior, colocando mais um campo booleano com o nome bloqueado, que bloqueia as contas com saldo negativo. Altere os métodos relacionados para que os mesmos devolvam mensagens (por exemplo 1, se foi possível sacar e 0 se não foi possível). Você também deve controlar na classe o número de clientes bloqueados (para isto utilize um atributo static). Teste o funcionamento da nova classe com o Junit.
3. Altere a classe **Cliente** para **Client3** de forma que existam no sistema no máximo cinco clientes simultaneamente. Para isso, faça com que os construtores sejam privados (isso é, sem acesso externo) e crie um método público estático **criaCliente** que devolva objetos do tipo **Cliente**; se houver mais de 5 clientes, seu método deve retornar **null**. Por que o método **criaCliente** deve ser estático? (escreva um comentário antes deste método com sua explicação)

Crie também o método `finalize()` para a classe `Cliente3`, de tal forma que quando um objeto `Cliente3` não é mais referenciado um novo objeto `Cliente3` possa ser criado<sup>1</sup>. Teste o funcionamento com JUnit.

4. Verifique quantos objetos são criados antes do coletor de lixo ser chamado para diversos tamanhos de vetor para o exemplo abaixo:

```
public class OcupaMemoria {
    static int quantos = 0;
    static boolean finalizou = false;
    double a[] = new double[100]; // apenas para ocupar espaco
    public OcupaMemoria(){
        quantos++;
    }
    protected void finalize() {
        if (!finalizou){
            System.out.println("Finalizou uma vez após criar "+
                               quantos+" objetos");
            finalizou = true; // não imprime mais mensagens
        }
    }
    public static void teste(){
        while (OcupaMemoria.finalizou==false)
            new OcupaMemoria();
    }
}
```

Como você explica o funcionamento deste programa? Verifique o que o `System.gc()` faz e use isto no programa. Rode várias vezes e veja o que acontece. Mude o valor do vetor `a[]` e veja o que acontece. Deixe sua explicação como um comentário no início do programa.

5. Conforme visto em aula, existe uma diferença de velocidade conforme o acesso a diferentes regiões da memória, verifique isto criando variações do programa abaixo:

```
import java.util.*;
import java.lang.*;

public class TesteTempo {
    private final static int TAMANHO = 100000;
    private final static int MAXIT = 10;
    private static int[] vint = new int[TAMANHO];

    public static void preenche() {
```

---

<sup>1</sup>Dica: Para que o método `finalize()` seja efetivamente chamado crie um método `fimCliente()` que o chama.

```

        for (int i = 0; i < TAMANHO; i++) {
            vint[i] = i;
        }
    }

    public static long testeint(int i) {
        long y = 0;
        long inicio = System.currentTimeMillis();
        for (int k = 0; k < MAXIT; k++)
            for (int j = 0; j < TAMANHO; j++) {
                y += vint[j]; // atribui a y a soma de 1 a TAMANHO-1
            }
        long fim = System.currentTimeMillis();
        System.out.println("int, teste:" +
                           i + ":Tempo gasto: " + (fim - inicio) + "ms");
        return (fim - inicio);
    }

    public static void main(String[] args) {
        long min, med, soma;
        long aux;
        min = Integer.MAX_VALUE;
        soma = 0;
        preenche();
        System.out.println("Teste para 5 iteracoes");
        for (int i = 0; i < 5; i++) {
            aux = TesteTempo.testeint(i + 1);
            // os testes devem ser feitos um apos o outro 5 vezes
            if (aux < min)
                min = aux;
            soma += aux;
        }
        med = soma / 5;
        System.out.println("Resultados finais: tempo minimo = " +
                           min + " tempo medio = " + med);
    }
}

```

Nas variações você deve fazer com que o tipo variável, ou objeto, do vetor (v) seja dos seguintes tipos: Integer, uma classe Inteiro com um inteiro público, e BigInteger. Para ter uma melhor estimativa do tempo, a cada iteração, você deve medir os tempos da soma com cada tipo de objeto. Conforme a classe utilizada você vai ter que utilizar métodos da mesma. Explique os resultados obtidos como um comentário no início do programa.