

MÓDULO 1 - EXTRA CONCURRENCY



**COMO TESTAR NOSSO
SERVIDOR COM MUITOS
CLIENTES SIMULTÂNEOS?**



- **Ferramenta veterana:** com ampla adoção no mercado tem um ecossistema de plugins muito rico
- **Interface Gráfica (GUI):** um grande facilitador para quem está começando com testes de carga
- **Integração com Java:** escrito em Java e pode ser estendido com seus próprios scripts Java ou Groovy
- **Simulação de Cenários Complexos:** permite simular a interação do usuário, com lógica condicional, processamento de dados de resposta e encadeamento de requisições.

Download de um zip (ou sdk `install jmeter`), execução com Java

<https://jmeter.apache.org/>

Configuração do JMeter

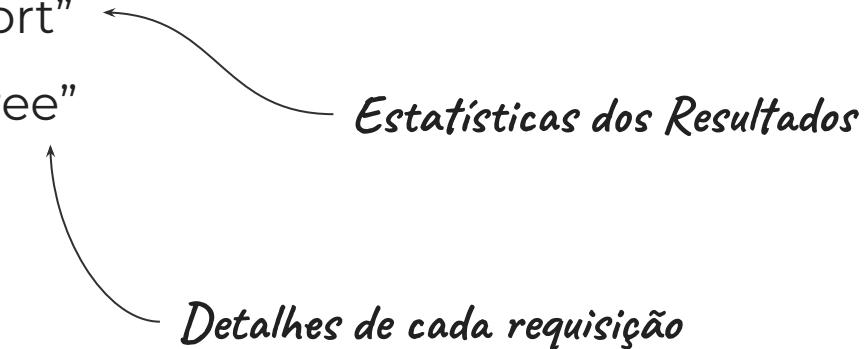
- Criar “Test Plan” chamado **Teste Servidor Cardápio Socket**
- Adicionar “Thread Group” chamado **Acessando Cardápio**
 - Número de Threads (users): **100**
 - Ramp-up periodo (seconds): **10**
 - Loop count: **Infinite**
 - Specify Thread lifetime: **Sim**
 - Duration (seconds): **60**

Simula usuários simultâneos

Baseado em duração, não em contagem de requests
- Adicionar “Sampler > Http Request” chamado **GET Itens Cardápio JSON**
 - Protocol: **http**
 - Server Name or IP: **localhost**
 - Port Number: **8000**
 - HTTP Request: **GET**
 - Path: **/itensCardapio.json**

Configuração do JMeter (continuação)

- Adicionar “Listener > Summary Report”
- Adicionar “Listener > View Results Tree”
- Salvar arquivo .jmx



Simulando demora ao tratar requisição

```
//...  
String request = requestBuilder.toString();  
System.out.println(request);
```

```
Thread.sleep(250);
```

*Adiciona pequena espera (250 ms)
para simular processamento*

```
Path path = Path.of("itensCardapio.json");  
String json = Files.readString(path);  
//...
```

Resultados do JMeter

- Requisições totais: **333**
- Tempo de resposta mínimo: **274 ms**
- Tempo de resposta médio: **21.2 s**
- Tempo de resposta máximo: **81.5 s**
- Vazão: **3.63 reqs/sec**

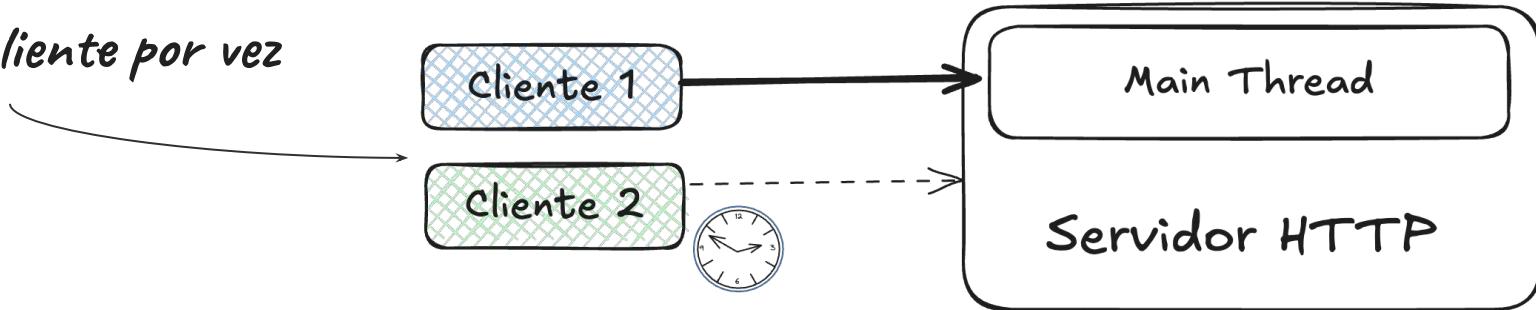
The screenshot shows the JMeter interface with the following details:

- Toolbar:** Includes icons for file operations, test plan management, and monitoring.
- Test Plan Tree:** Shows a tree structure with "Teste Servidor Cardápio Socket" expanded, containing "Acessando Cardápio" and "Summary Report".
- Summary Report Configuration:** A dialog box titled "Summary Report" with fields:
 - Name: Summary Report
 - Comments: (empty)
 - Write results to file / Read from file
 - Filename: (empty)
 - Browse... button
 - Log/Display Only: Errors Successes Configure
- Results Table:** A table showing the summary statistics for the "GET Itens Cardápio JSON" sampler.

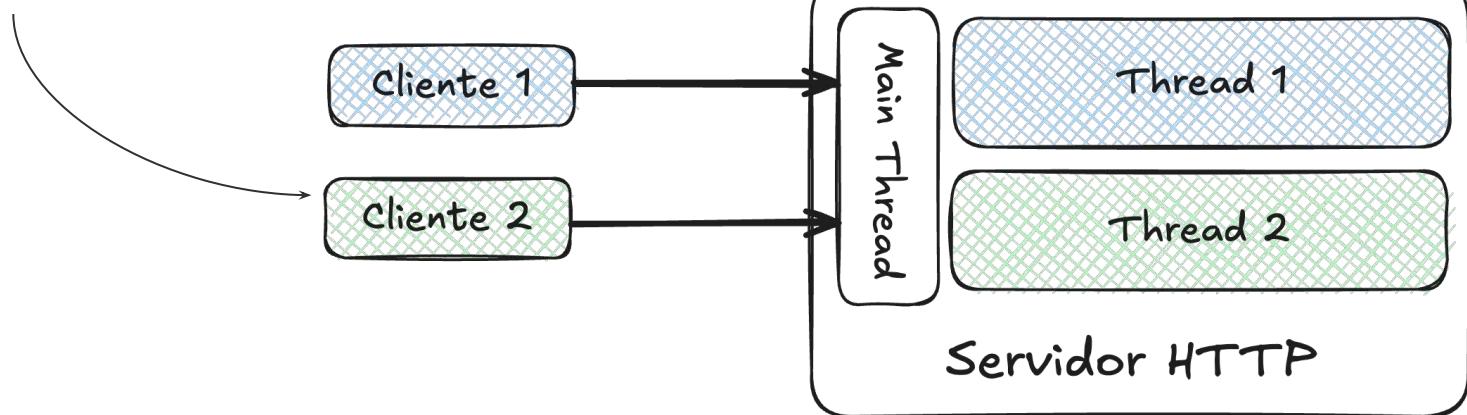
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET Itens Cardápio JSON	333	21242	276	81502	23340.07	0.00%	3.6/sec	3.90	0.48	1100.0
TOTAL	333	21242	276	81502	23340.07	0.00%	3.6/sec	3.90	0.48	1100.0
- Status Bar:** Shows "00:01:31" and other metrics.

Multi-threading

Apenas um cliente por vez

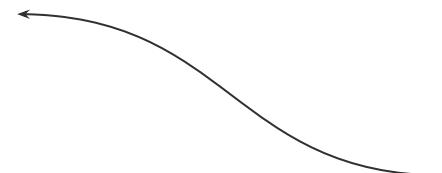


Vários clientes ao mesmo tempo



Organizando código

```
try(ServerSocket serverSocket = new ServerSocket(8000)) {  
    System.out.println("Subiu servidor!");  
  
    while (true) {  
        Socket clientSocket = serverSocket.accept();  
        trataRequisicao(clientSocket);  
    }  
}
```



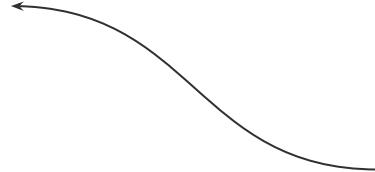
*Extraindo método para lidar com
request antes de continuar*

Organizando código (continuação)

```
private static void trataRequisicao(Socket clientSocket) {  
    try (clientSocket) { ← try-with-resources aqui  
        InputStream clientIS = clientSocket.getInputStream();  
        StringBuilder requestBuilder = new StringBuilder();  
        //...  
        clientOut.println();  
        clientOut.println(json);  
  
    } catch (Exception ex) { ← Embrulhando checked exception  
em uma RuntimeException  
        throw new RuntimeException(ex);  
    }  
}
```

Iniciando um thread por requisição

```
try(ServerSocket serverSocket = new ServerSocket(8000)) {  
    System.out.println("Subiu servidor!");  
  
    while (true) {  
        Socket clientSocket = serverSocket.accept();  
        new Thread(() -> trataRequisicao(clientSocket)).start();  
    }  
}
```



*Criando e iniciando uma nova
Thread a cada nova requisição*

Resultados do JMeter com threads

- Requisições totais: **10807** ↪ *32x mais requisições*
- Tempo de resposta mínimo: **501 ms** → *2x maior*
- Tempo de resposta médio: **511 ms** ↪ *41x menor*
- Tempo de resposta máximo: **555 ms** ↪ *146x menor*
- Vazão: **178.6 reqs/sec** ↪ *49x maior*

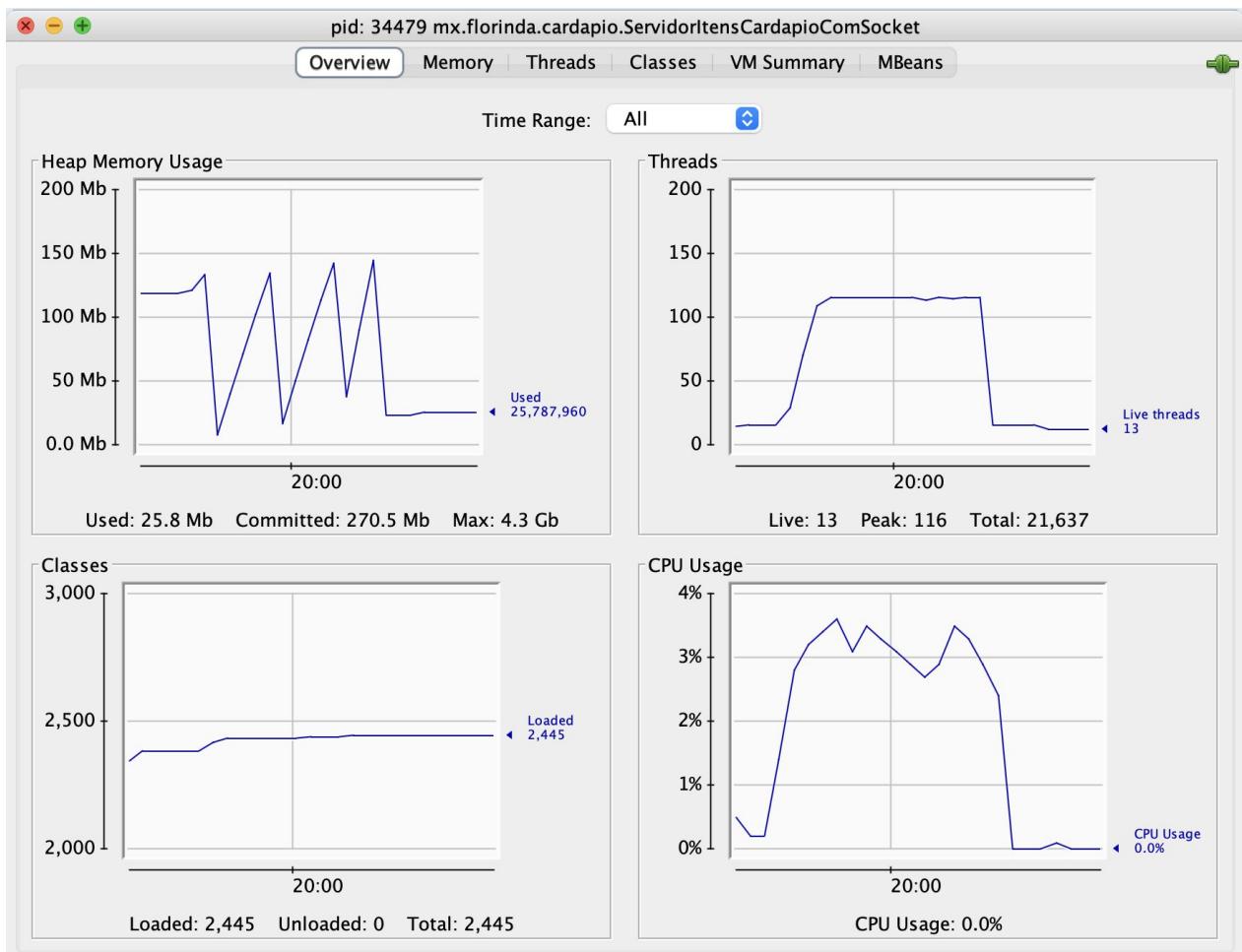
The screenshot shows the JMeter interface with the 'Summary Report' dialog open. The left sidebar shows a tree structure with 'Teste Servidor Cardápio Socket' expanded, containing 'Acessando Cardápio', 'GET Itens Cardápio JSON', 'View Results Tree', and 'Summary Report'. The 'Summary Report' item is selected. The main area displays the 'Summary Report' configuration and a detailed table of results.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET Itens Cardápio JSON	10807	511	501	555	4.75	0.00%	178.6/sec	191.86	23.37	1100.0
TOTAL	10807	511	501	555	4.75	0.00%	178.6/sec	191.86	23.37	1100.0

jconsole

- max 116 threads
- max 3.5% de CPU
- max 150 MB RAM

Não ter um limite para o número de threads por sobrecarregar o servidor



Thread pool com Executor

Thread pool fixo com
até 50 threads

```
Executor executor = Executors.newFixedThreadPool(50);
```

```
try(ServerSocket serverSocket = new ServerSocket(8000)) {  
    System.out.println("Subiu servidor!");  
  
    while (true) {  
        Socket clientSocket = serverSocket.accept();  
        executor.execute(() -> trataRequisicao(clientSocket));  
    }  
}
```

Invocando Executor a cada
nova requisição

Resultados do JMeter com thread pool

- Requisições totais: **5729** ← 1.9x menos que sem pool, 173x mais que sem threads
- Tempo de resposta mínimo: **503 ms**
- Tempo de resposta médio: **968 ms** ← 1.9x maior que sem pool
- Tempo de resposta máximo: **1038 ms** ← 1.9x maior que sem pool
- Vazão: **94.0 reqs/sec** ← 1.9x menor que sem pool, 26x maior que sem threads

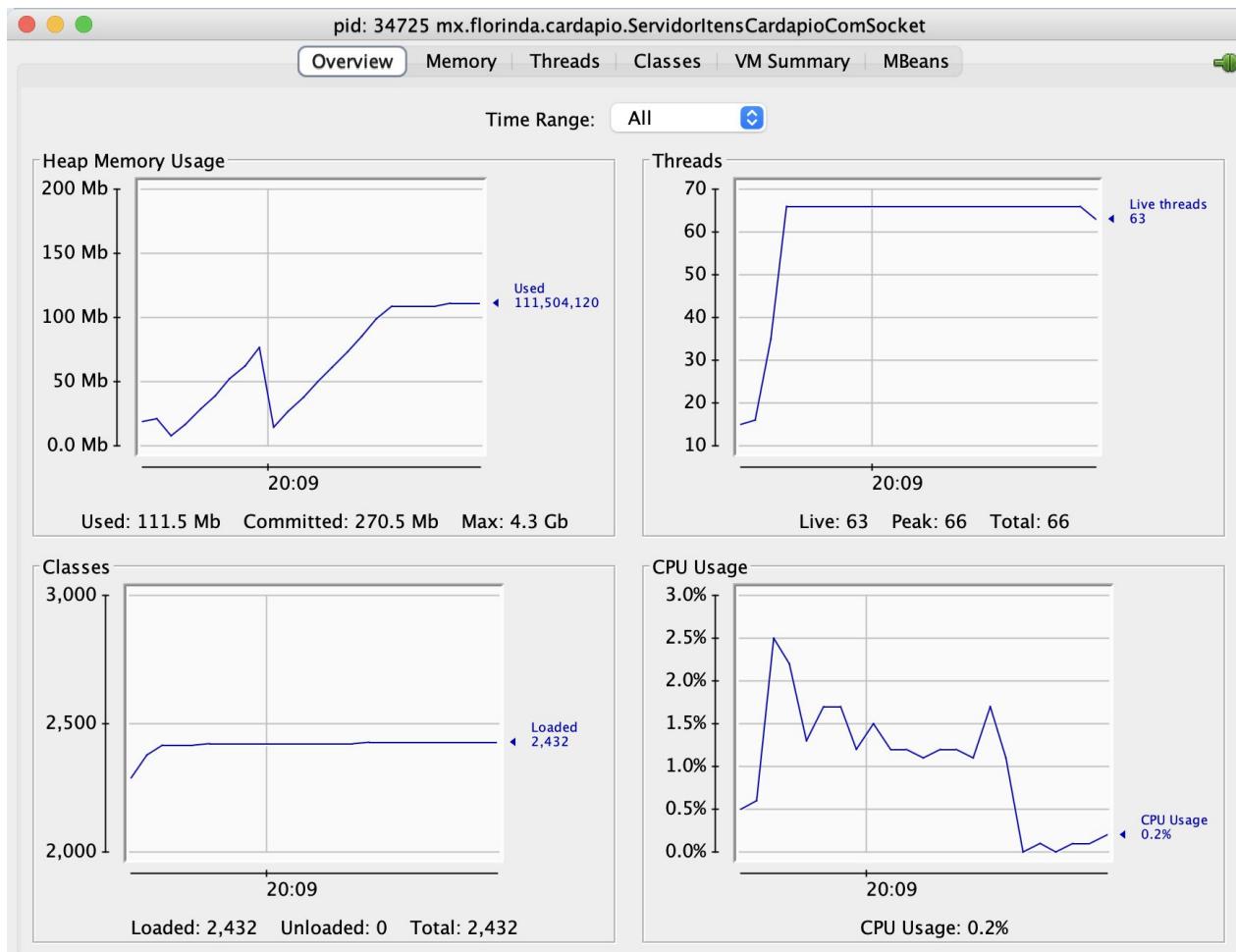
The screenshot shows the JMeter interface with the 'Summary Report' configuration selected in the left sidebar. The main area displays the 'Summary Report' settings, including fields for 'Name' (Summary Report), 'Comments', 'Write results to file / Read from file', 'Filename', 'Log/Display Only' checkboxes for 'Errors' and 'Successes', and a 'Configure' button. Below this, a detailed results table is shown:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET Itens Cardá...	5729	968	503	1038	133.57	0.00%	94.0/sec	100.93	12.29	1100.0
TOTAL	5729	968	503	1038	133.57	0.00%	94.0/sec	100.93	12.29	1100.0

jconsole 2

- max 66 threads
- max 2.5% de CPU
- max 111 MB RAM

Número máximo de threads fica limitado



EXERCÍCIO PRÁTICO

Listagem, quantidade e adição de Itens de Cardápio na API.

Implemente os seguintes endpoints no nosso Servidor HTTP com Socket usando a nossa classe Database como “banco de dados”.

- GET /itens-cardapio - Lista todos os itens do cardápio
- GET /itens-cardapio/total - Retorna a quantidade de itens do cardápio
- POST /itens-cardapio - Adiciona um item do cardápio recebido em um JSON

Teste usando o curl.

Obtendo infos do request

```
System.out.println(request);
System.out.println("\n\nChegou um novo request");

String[] requestChunks = request.split("\r\n\r\n");
String requestLineAndHeaders = requestChunks[0];
String[] requestLineAndHeadersChunks = requestLineAndHeaders.split("\r\n");

String requestLine = requestLineAndHeadersChunks[0];
String[] requestLineChunks = requestLine.split(" ");
String method = requestLineChunks[0];
String requestURI = requestLineChunks[1];
String httpVersion = requestLineChunks[2];

System.out.println("Method: " + method);
System.out.println("Request URI: " + requestURI);
System.out.println("HTTP Version: " + httpVersion);
```

*Extraindo informações
da requisição*

Tratando URIs

```
OutputStream clientOS = clientSocket.getOutputStream();
PrintStream clientOut = new PrintStream(clientOS);

if ("/itensCardapio.json".equals(requestURI)) {
    System.out.println("Chamou arquivo itensCardapio.json");
    // ...
} else {
    System.out.println("URI não encontrada: " + requestURI);
    clientOut.println("HTTP/1.1 404 Not Found");
}
```

Trata o caso de URI não encontrada

Listagem de Itens do Cardápio

Define atributo para
o Database

```
private static final Database database = new Database();  
  
} else if ("GET".equals(method) && "/itens-cardapio".equals(requestURI)) {  
    System.out.println("Chamou listagem de itens de cardápio");  
    List<ItemCardapio> listaItensCardapio = database.listaItensCardapio();  
  
    Gson gson = new Gson();  
    String json = gson.toJson(listaItensCardapio);  
  
    clientOut.println("HTTP/1.1 200 OK");  
    clientOut.println("Content-type: application/json; charset=UTF-8");  
    clientOut.println();  
    clientOut.println(json);  
}
```

Obtem listagem do Database

```
curl -v localhost:8000/itens-cardapio
```

Total de Itens do Cardápio

```
public int totalItensCardapio() {  
    return itensPorId.size();  
}  
  
Método adicionado ao Database }  
  
}  
} else if ("GET".equals(method) && "/itens-cardapio/total".equals(requestURI)) {  
  
    System.out.println("Chamou total de itens de cardápio");  
    int totalItens = database.totalItensCardapio(); ←  
  
    clientOut.println("HTTP/1.1 200 OK");  
    clientOut.println();  
    clientOut.println(totalItens);  
}  
}
```

Obtem total do Database

```
curl -v localhost:8000/itens-cardapio/total
```

Adição de novo Item do Cardápio

Método adicionado ao
Database

```
public void adicionaItemCardapio(ItemCardapio item) {  
    itensPorId.put(item.id(), item);  
}  
  
else if ("POST".equals(method) && "/itens-cardapio".equals(requestURI)) {  
    System.out.println("Chamou adição de itens de cardápio");  
    if (requestChunks.length == 1) {  
        clientOut.println("HTTP/1.1 400 Bad Request");  
    }  
    String body = requestChunks[1];  
    Gson gson = new Gson();  
    ItemCardapio item = gson.fromJson(body, ItemCardapio.class);  
    database.adicionaItemCardapio(item);  
    clientOut.println("HTTP/1.1 200 OK");  
}
```

Adiciona item no Database

Testando adição de Item do Cardápio

```
curl -v
```

```
-X POST
```

```
-H 'application/json'
```

```
-d '{"id":1,"nome":"Item 1",  
     "descricao":"Item 1",  
     "categoria":"PRATOS_PRINCIPAIS",  
     "preco":3.9,  
     "precoPromocional":2.99}'
```

Tudo na mesma linha

```
localhost:8000/itens-cardapio
```

TESTANDO ADIÇÃO EM ESCALA COM O JMETER

Alterando Configuração do JMeter

- Mudar “Número de Threads” em Acessando Cardápio para: **20**
- Desabilitar GET Itens Cardápio JSON
- Adicionar “**Sampler > Http Request**” chamado **POST Itens Cardápio**
 - Protocol: **http**
 - Server Name or IP: **localhost**
 - Port Number: **8000**
 - HTTP Request: **POST**
 - Path: **/itens-cardapio**
 - Body Data: `{"id":${contador}, "nome": "Item ${contador}", "descricao": "Item ${contador}", "categoria": "PRATOS_PRINCIPAIS", "preco":3.9, "precoPromocional":2.99}`
- Adicionar “**Config Element > Counter**”
 - Starting Value: **1**
 - Increment: **1**
 - Exported Variable Name: **contador**

Preparando teste de adição com JMeter

```
public Database() {  
    //         var refrescoDoChaves = new ItemCardapio(1L, "Refresco do Chaves",  
    //                                         "Suco de limão que parece de tamarindo e tem gosto de groselha.",  
    //                                         BEBIDAS, new BigDecimal("2.99"), null);  
    //         itensPorId.put(refrescoDoChaves.id(), refrescoDoChaves);  
    //  
    //         var sanduicheDoChaves = new ItemCardapio(2L, "Sanduíche de Presunto do Chaves",  
    //                                         "Sanduíche de presunto simples, mas feito com muito amor.",  
    //                                         PRATOS_PRINCIPAIS, new BigDecimal("3.50"), new BigDecimal("2.99"));  
    //         itensPorId.put(sanduicheDoChaves.id(), sanduicheDoChaves);  
    // ...  
}
```



Comentando itens iniciais

Resultados do teste de adição com JMeter

The screenshot shows the JMeter interface with a tree view on the left and a 'Summary Report' dialog on the right. The tree view has a node 'Teste Servidor Cardápio Socket' expanded, showing 'Acessando Cardápio' and several test plans. The 'Summary Report' dialog is open, with 'Name' set to 'Summary Report' and 'Comments' empty. It includes options for writing results to file or reading from file, and a 'Browse...' button. Below these are checkboxes for 'Log/Display Only' (Errors, Successes) and a 'Configure' button. A table below the controls shows performance metrics for two requests: 'POST Itens Cardápio' and a total row. The table has columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
POST Itens Card...	4332	255	251	355	2.43	0.00%	72.0/sec	1.12	22.93	16.0
TOTAL	4332	255	251	355	2.43	0.00%	72.0/sec	1.12	22.93	16.0

4332 itens adicionados

```
$ curl localhost:8000/itens-cardapio/total
```

4264

68 itens a menos

Onde foram parar esses itens? 🤔

THREAD-SAFE COLLECTIONS

Não

Thread-safe

Thread-safe
mas ineficiente

Thread-safe
e eficiente

ArrayList
LinkedList

Vector
Collections
.synchronizedList

CopyOnWriteArrayList

HashSet
LinkedHashSet
TreeSet

Collections
.synchronizedSet

CopyOnWriteHashSet
ConcurrentSkipListSet
ordenado

HashMap
LinkedHashMap
TreeMap

Hashtable
Collections
.synchronizedMap

ConcurrentHashMap
ConcurrentSkipListMap
ordenado

IdentityHashMap
WeakHashMap

Usando ConcurrentHashMap

```
import java.util.concurrent.ConcurrentHashMap;

public class Database {

    private final Map<Long, ItemCardapio> itensPorId = new ConcurrentHashMap<>();

    // ...

}
```



*Versão Thread-safe e
eficiente do HashMap*

Resultados do teste com JMeter

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
POST Itens Card...	4277	258	251	359	4.52	0.00%	71.0/sec	1.11	22.62	16.0
TOTAL	4277	258	251	359	4.52	0.00%	71.0/sec	1.11	22.62	16.0

4227 itens adicionados
(105 itens a menos, vazão um pouco menor)

```
$ curl localhost:8000/itens-cardapio/total  
4277
```

Mesma quantidade! 😊

Usando ConcurrentSkipListMap

```
import java.util.concurrent.ConcurrentSkipListMap;  
  
public class Database {  
  
    private final Map<Long, ItemCardapio> itensPorId = new ConcurrentSkipListMap<>();  
  
    // ...  
}
```



*Versão Thread-safe do
TreeMap, eficiente e que
mantém ordenado pelas
chaves*

Resultados do teste com JMeter

The screenshot shows the JMeter interface with a tree view on the left containing 'Teste Servidor Cardápio Socket' and its sub-items: 'Acessando Cardápio', 'GET Itens Cardápio JSON', 'View Results Tree', 'Summary Report' (which is selected), 'POST Itens Cardápio', and 'Counter'. The main panel displays the 'Summary Report' dialog with fields for 'Name' (Summary Report), 'Comments', 'Write results to file / Read from file', 'Filename' (empty), 'Log/Display Only' (checkboxes for Errors and Successes), and 'Configure'. Below the dialog is a table with the following data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
POST Itens Card...	4272	258	251	364	4.08	0.00%	71.1/sec	1.11	22.66	16.0
TOTAL	4272	258	251	364	4.08	0.00%	71.1/sec	1.11	22.66	16.0

4272 itens adicionados

```
$ curl localhost:8000/itens-cardapio/total  
4272
```

Ordenação certinha!

```
$ curl localhost:8000/itens-cardapio  
..., {"id":4271, "nome":"Item 4271", "descricao":"Item 4271",  
"categoria":"PRATOS_PRINCIPAIS", "preco":3.9, "precoPromocional":2.99},  
{"id":4272, "nome":"Item 4272", "descricao":"Item 4272",  
"categoria":"PRATOS_PRINCIPAIS", "preco":3.9, "precoPromocional":2.99}]
```