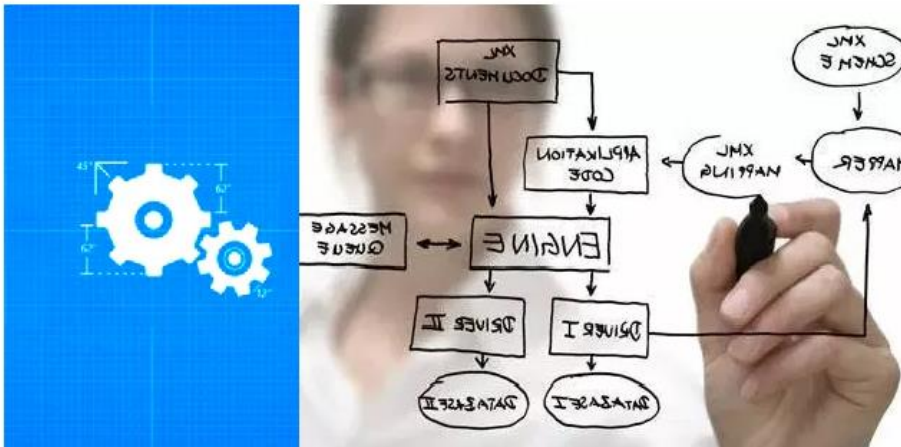


Arquitetura de Software

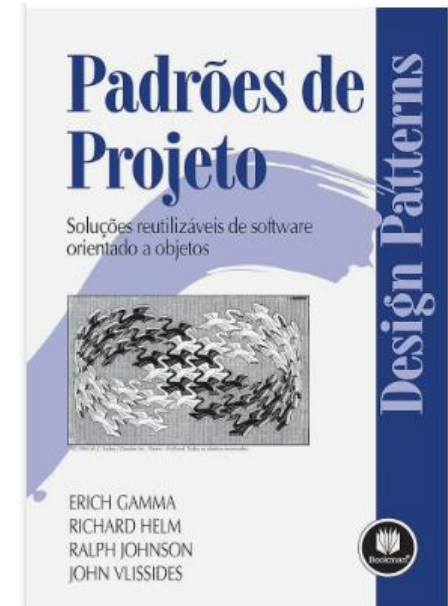
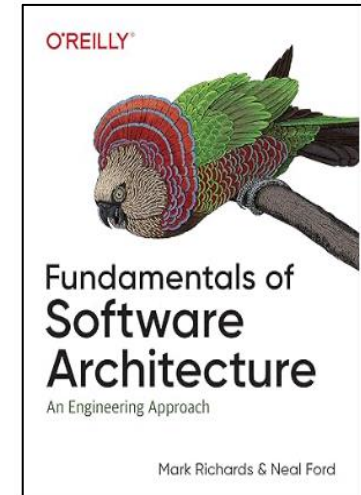
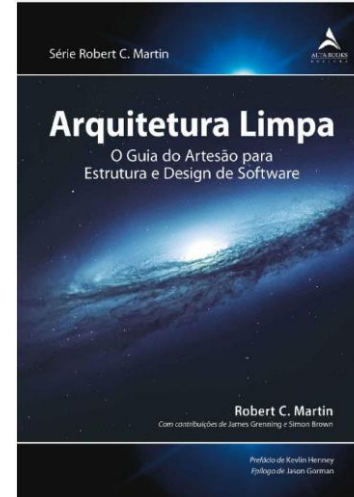
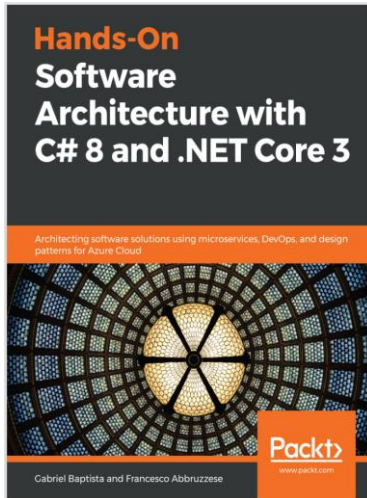
Unidade 2 - Introdução à Arquitetura de Software

Arquitetura Monolítica



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

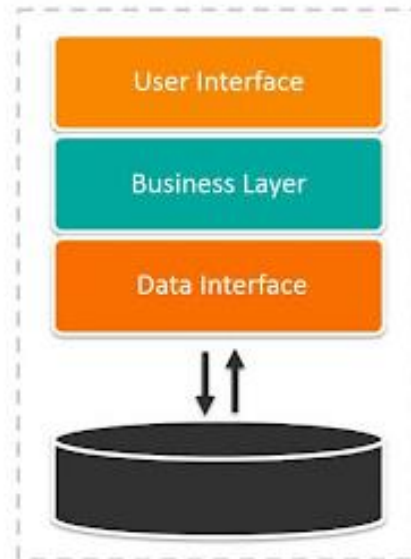
Bibliografia



Arquitetura Monolítica

- ✚ Em uma **arquitetura monolítica**, todos os componentes do software (interface do usuário, lógica de negócios, acesso a dados, etc.) são integrados em um **único programa**, ou "**monólito**";
- ✚ Essa abordagem trata a aplicação como uma **unidade indivisível** e atômica.

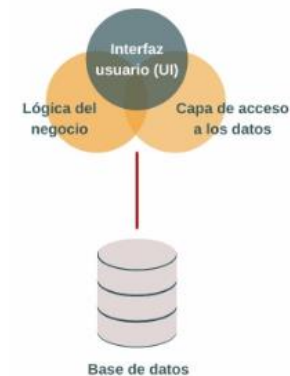
Monolithic Architecture



Arquitetura Monolítica – Características

- **Integração Tightly Coupled:** Todos os componentes e funcionalidades estão fortemente interligados e operam como uma única unidade.
- **Simplicidade de Desenvolvimento e Implantação:** Inicialmente, é mais simples de desenvolver e implantar, pois tudo é centralizado.
- **Escalabilidade Horizontal:** Para escalar uma aplicação monolítica, geralmente replicamos o monólito inteiro.

Arquitetura monolítica





Uma aplicação mainframe da década de 80 emprega arquitetura monolítica ?

Arquitetura Monolítica

- ⊕ **Sim**, os sistemas de **mainframe** da década de 1980 são bons exemplos de arquitetura monolítica.



Sistemas de Mainframe da Década de 1980

Características

1. **Centralização:** Os mainframes eram sistemas centralizados que executavam todas as operações de computação, armazenamento e processamento de dados em uma única máquina.
2. **Software e Hardware Integrados:** O software desenvolvido para esses sistemas era frequentemente específico para o hardware do mainframe e funcionava como um único sistema integrado.
3. **Aplicações Empresariais:** Eram comumente usados para aplicações empresariais críticas, como processamento de transações bancárias, sistemas de folha de pagamento e gerenciamento de registros.



Porque uma aplicação mainframe da década de 80 é um exemplo de arquitetura monolítica ?

Mainframe – Década de 80

Por que são Exemplos de Arquitetura Monolítica?

- **Software Unificado:** Os aplicativos eram muitas vezes grandes programas unificados, onde diferentes funções e módulos estavam todos contidos dentro de uma única base de código e ambiente de execução.
- **Manutenção e Atualização:** Atualizar ou modificar uma parte do sistema muitas vezes significava ter que reimplantar ou reiniciar o sistema inteiro.
- **Escalabilidade e Flexibilidade Limitadas:** Devido à sua natureza integrada, escalar ou adaptar esses sistemas para novas necessidades era um processo complexo e muitas vezes dispendioso.



Mainframe – Década de 80

Contextualização Histórica

- **Tecnologia da Época:** Na década de 1980, a tecnologia e as práticas de desenvolvimento de software eram muito diferentes das atuais. A modularidade e os microsserviços ainda não eram conceitos amplamente adotados ou viáveis, especialmente dadas as limitações de hardware e rede da época.
- **Evolução dos Mainframes:** Embora os mainframes modernos tenham evoluído e agora possam suportar uma variedade de arquiteturas de software, os sistemas de mainframe mais antigos são representativos do paradigma monolítico.



Uma aplicação desktop desenvolvida em C pode ser um exemplo de arquitetura monolítica ?

Arquitetura Monolítica

- ⊕ **Sim**, uma aplicação desktop desenvolvida em **C** pode ser um exemplo de **arquitetura monolítica**.



Aplicação Desktop em C

Aplicação Desktop em C com Arquitetura Monolítica

Características da Aplicação

1. **Desenvolvimento Integrado:** A aplicação é desenvolvida como um único código-fonte em C, onde todas as funcionalidades, como interface do usuário, lógica de negócios e manipulação de dados, estão integradas.
2. **Compilação e Execução:** O código é compilado em um único executável. Quando a aplicação é executada, todas as suas funcionalidades são carregadas em um único processo.
3. **Exemplo de Uso:** Um editor de texto ou uma ferramenta de gerenciamento financeiro pessoal que roda localmente no computador do usuário.

Aplicação Desktop em C

Por que é Monolítica?

- **Unidade Única de Implantação:** Todos os aspectos da aplicação, desde a interface até a lógica de processamento, são parte de um único programa executável.
- **Manutenção e Atualização:** Atualizar qualquer parte da aplicação geralmente requer recompilação e redistribuição do executável completo.
- **Escalabilidade e Flexibilidade:** Como um monólito, mudanças significativas ou a escalabilidade da aplicação podem ser mais desafiadoras, pois qualquer alteração afeta o sistema inteiro.

Aplicação Desktop em C

Contexto Típico

- **Aplicações de Escopo Limitado:** Arquiteturas monolíticas são comuns em aplicações desktop onde a complexidade é gerenciável e as vantagens de uma arquitetura mais distribuída, como microsserviços, não são necessárias.
- **Linguagem C:** A linguagem C é frequentemente escolhida por sua eficiência e controle próximo ao hardware, o que pode ser vantajoso para aplicações desktop que requerem alto desempenho.

Aplicação Desktop em C

Portanto, uma aplicação desktop desenvolvida em C pode ser um bom exemplo de uma arquitetura monolítica, especialmente se for projetada como um sistema unificado e autocontido. Essa abordagem pode ser adequada para aplicações que não exigem a complexidade de sistemas distribuídos e se beneficiam da simplicidade e da eficiência de uma arquitetura monolítica.





Uma aplicação desktop que faz backup local em um PC de um usuário, pode ser um exemplo de arquitetura monolítica ?

Arquitetura Monolítica

- ✦ **Sim**, uma aplicação desktop que faz **backup local** em um PC de um usuário, pode ser um exemplo de **arquitetura monolítica**.



Backup local

1. **Funcionalidade Integrada:** O aplicativo gerencia tarefas como identificar arquivos para backup, copiá-los para um local seguro (como um disco rígido externo ou uma partição diferente), e possivelmente comprimi-los ou criptografá-los. Todas essas funções estão integradas em um único software.
2. **Execução Unificada:** Quando o usuário executa o aplicativo, todas as suas funções são carregadas e executadas como parte de um único processo ou serviço.
3. **Exemplo de Uso:** Um software simples de backup que um usuário pode configurar para fazer backups automáticos de documentos importantes.

Backup local

Por que é Monolítica?

- **Unidade Única de Implantação:** O software é desenvolvido, compilado (se necessário) e implantado como um único executável ou pacote de software.
- **Manutenção e Atualização:** Qualquer mudança, seja na lógica de backup, na interface do usuário ou em outras funcionalidades, requer a recompilação e a reinstalação do programa inteiro.
- **Escalabilidade e Flexibilidade:** Embora a escalabilidade possa não ser uma grande preocupação para aplicações de backup local, a arquitetura monolítica significa que todas as funcionalidades estão entrelaçadas, o que pode tornar as atualizações e manutenções mais desafiadoras.

Backup local

Vantagens e Desvantagens

- **Vantagens:** Simplicidade de desenvolvimento e instalação, bem como a facilidade de uso para o usuário final.
- **Desvantagens:** Limitações em termos de escalabilidade e modularidade; atualizações e manutenção podem exigir a manipulação do aplicativo inteiro.

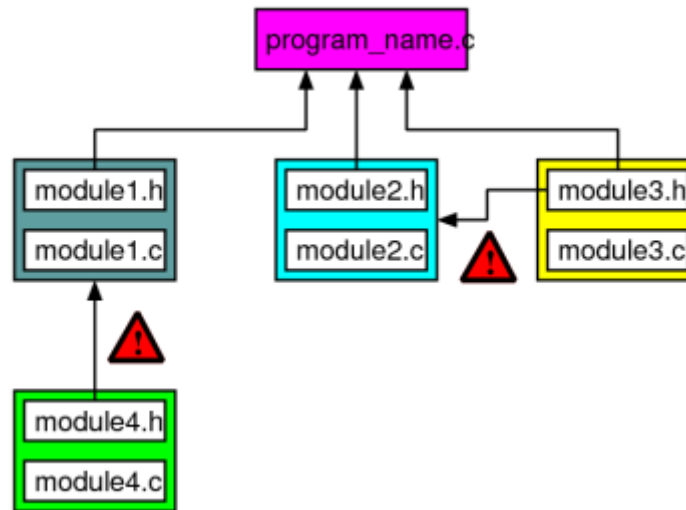




Uma aplicação desktop desenvolvida em C que use diversas funções, sendo estruturada em módulos, pode ser um exemplo de arquitetura monolítica ?

Arquitetura Monolítica

- ⊕ **Sim**, uma aplicação **desktop** desenvolvida em **C** que use diversas funções, **sendo estruturada em módulos**, pode ser um exemplo de arquitetura monolítica.



Aplicação Desktop em C com Estrutura Modular

Estrutura da Aplicação

- **Módulos:** A aplicação pode ser organizada em diferentes módulos ou arquivos de código-fonte em C, cada um responsável por uma funcionalidade específica (por exemplo, processamento de dados, interface do usuário, operações de arquivo).
- **Compilação:** Apesar da estrutura modular no código-fonte, todos os módulos são compilados juntos para formar um único executável.

Aplicação Desktop em C com Estrutura Modular

Características Monolíticas

- **Execução como uma Unidade:** Quando a aplicação é executada, ela opera como um único processo, com todos os módulos carregados juntos.
- **Implantação Unificada:** A aplicação é distribuída e implantada como um único executável, independentemente de sua estrutura interna modular.
- **Manutenção e Atualização:** Atualizações em qualquer parte da aplicação geralmente exigem a recompilação e a redistribuição do executável completo.

Aplicação Desktop em C com Estrutura Modular

Por Que Ainda é Monolítica?

- **Integração Tightly Coupled:** Apesar da organização em módulos, a aplicação é fortemente acoplada na medida em que todos os módulos são parte integrante do único executável.
- **Independência e Escalabilidade:** A estrutura modular interna não altera o fato de que a aplicação é executada, gerenciada e escalada como uma entidade única.

Aplicação Desktop em C com Estrutura Modular

Diferença entre Modularidade e Arquitetura

- **Modularidade:** Refere-se à organização do código-fonte e à separação de responsabilidades dentro do software. Uma aplicação pode ser modular em seu design interno, mas ainda ser monolítica em termos de execução e implantação.
- **Arquitetura Monolítica:** Um conceito mais amplo que se refere a como o software é construído, implantado e executado como um todo.

Conclusão

Portanto, uma aplicação desktop em C com uma estrutura modular ainda representa uma arquitetura monolítica se todas as suas partes forem compiladas em um único executável e operarem como um único processo. A modularidade interna ajuda na organização e manutenção do código, mas não altera a natureza monolítica da arquitetura da aplicação.



Uma aplicação Java WEB servlet com JSP com acesso a um servidor de banco de dados pode ser um exemplo de arquitetura monolítica ?

Aplicação Java WEB servlet com JSP com acesso a um servidor de banco de dados

- ⊕ **Sim**, uma **Java WEB servlet** com **JSP** com acesso a um servidor de banco de dados pode ser um exemplo de **arquitetura monolítica**;
- ⊕ Esta **configuração** também se encaixa no modelo **cliente-servidor**, por se tratar de uma aplicação **WEB**.

- JDK (Java Development Kit)
- Eclipse IDE Enterprise Edition
- Apache tomcat
- MySQL

JDK



Aplicação Java WEB servlet com JSP com acesso a um servidor de banco de dados

Arquitetura Monolítica

Características

- **Integração de Componentes:** Toda a lógica de negócios, processamento de dados, e geração de interfaces de usuário (páginas JSP) estão contidos em uma única aplicação. Apesar de haver separação interna (diferentes Servlets e JSPs), tudo é compilado e implantado como uma única unidade (geralmente um arquivo WAR) em um servidor de aplicação, como o Tomcat.
- **Manutenção e Atualização:** Qualquer alteração na aplicação, seja na lógica dos Servlets, no design das páginas JSP, ou na lógica de acesso ao banco de dados, requer a recompilação e reimplantação do pacote completo.

Exemplo

- **Sistema de Gerenciamento de Pedidos:** Uma aplicação web para gerenciar pedidos de clientes, incluindo interfaces para inserção de pedidos, visualização de status, e administração de produtos, com todas essas funcionalidades sendo processadas no mesmo sistema.

Aplicação Java WEB servlet com JSP com acesso a um servidor de banco de dados

Arquitetura Cliente-Servidor

Características

Cliente: Navegadores dos usuários que solicitam páginas web e interagem com a aplicação.

Servidor: O servidor de aplicação (como o Apache Tomcat) que hospeda a aplicação Servlet/JSP.

Servidor de Banco de Dados: Uma entidade separada para armazenamento de dados, acessada pela aplicação web para operações de banco de dados.

Exemplo

Aplicação E-commerce: Os usuários acessam o site para comprar produtos; a aplicação no servidor processa essas solicitações, interage com o banco de dados para transações e gerencia o inventário, e retorna as respostas apropriadas para o navegador do cliente.

Aplicação Java WEB servlet com JSP com acesso a um servidor de banco de dados

Conclusão

Portanto, uma aplicação Java Web com Servlets e JSP que acessa um banco de dados pode ser classificada como monolítica em relação à sua construção interna e implantação. Ao mesmo tempo, ela opera dentro de uma arquitetura cliente-servidor em termos de sua interação com os usuários (via navegador) e com o banco de dados. Essa dualidade é comum em muitas aplicações web, onde "monolítico" se refere à estrutura interna da aplicação, enquanto "cliente-servidor" descreve a maneira como ela interage com os clientes (navegadores) e outros sistemas (como bancos de dados).



Exemplo de uma aplicação **console** desenvolvida em **Java** com a Arquitetura Monolítica



Arquitetura Monolítica – Tarefa 2_01

```
1 var now = new Date();
2 var hours = now.getHours();
3 var minutes = now.getMinutes();
4 var seconds = now.getSeconds();
5
6 var ampm = "am";
7 var colon = '<IMG SRC="images/colon.gif">';
8
9 if (hours >= 12) {
10     ampm = "pm";
11     hours = hours - 12;
12 }
13
14 if (hours == 0) hours = 12;
15
16 if (hours < 10) hours = "0" + hours;
17 else hours = hours + '';
18
19 if (minutes < 10) minutes = "0" + minutes;
20 else minutes = minutes + '';
21
22 if (seconds < 10) seconds = "0" + seconds;
23 else seconds = seconds + '';
```



Código Java, com arquitetura **monolítica**, que cria uma lista com **10 perguntas** sobre as capitais de diferentes países;

A aplicação, **via console**, envia a pergunta e o usuário digita a resposta e envia para o programa. No final do quiz, são mostradas as respostas corretas, erradas e a nota do usuário (de 0 a 10). A aplicação é monolítica uma vez que qualquer alteração irá requerer recompilação de todo o código.

Arquitetura Monolítica – Tarefa 2_01

```
package br.uscs;

import java.util.Scanner;

public class Quiz_Capitais_1 {
    public static void main(String[] args) {
        String[][] capitaisPaíses = {
            { "Brasil", "Brasília" },
            { "França", "Paris" },
            { "Japão", "Tóquio" },
            { "Alemanha", "Berlim" },
            { "Canadá", "Ottawa" },
            { "Índia", "Nova Délhi" },
            { "Rússia", "Moscou" },
            { "Austrália", "Canberra" },
            { "Argentina", "Buenos Aires" },
            { "Egito", "Cairo" }
        };

        int score = 0;
        Scanner scanner = new Scanner(System.in);
```

Arquitetura Monolítica – Tarefa 2_01

```
for (int i = 0; i < capitaisPaises.length; i++) {  
    System.out.println("Qual é a capital do país " +  
        capitaisPaises[i][0] + "?");  
  
    String respostaUsuario = scanner.nextLine();  
  
    if (respostaUsuario.trim().equalsIgnoreCase(capitaisPaises[i][1])) {  
        System.out.println("Correto!");  
        score++;  
    } else {  
        System.out.println("Errado! A resposta correta é: " +  
            capitaisPaises[i][1]);  
    }  
}
```

Arquitetura Monolítica – Tarefa 2_01

```
scanner.close();  
System.out.println("\nO quiz terminou. Você acertou " +  
    score + " de " + capitaisPaises.length + " perguntas.");  
  
System.out.println("Sua nota é: " + (score * 10 / capitaisPaises.length));  
}  
}
```

Arquitetura Monolítica – Outro exemplo

ARQUITETURA MONOLÍTICA



Arquitetura Monolítica – Tarefa 2_02



Alterar o código **Java** para empregar **HashMap** ao invés de um **array** com 2 dimensões!

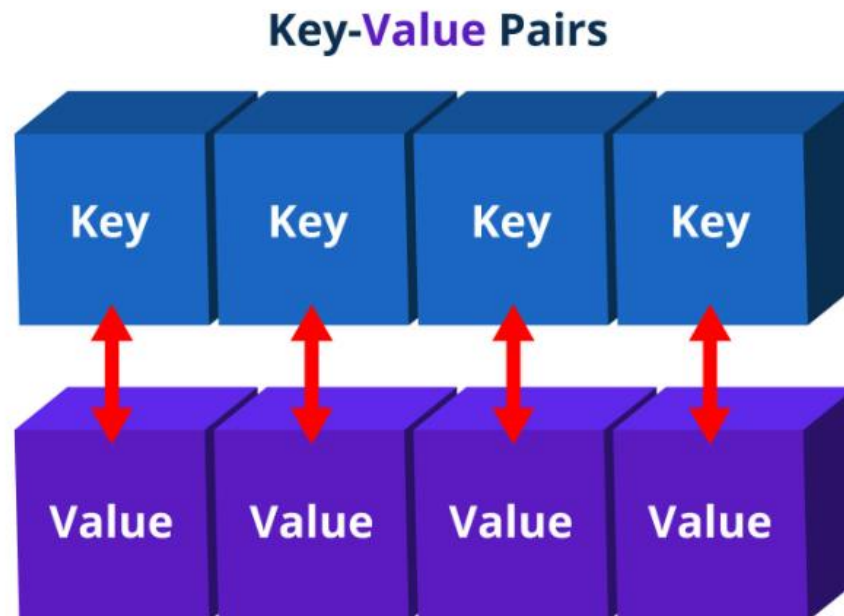
Como a arquitetura é **Monolítica**, **todo** o código deverá ser **recompilado**!



Mas, o que é HashMap ?

HashMap

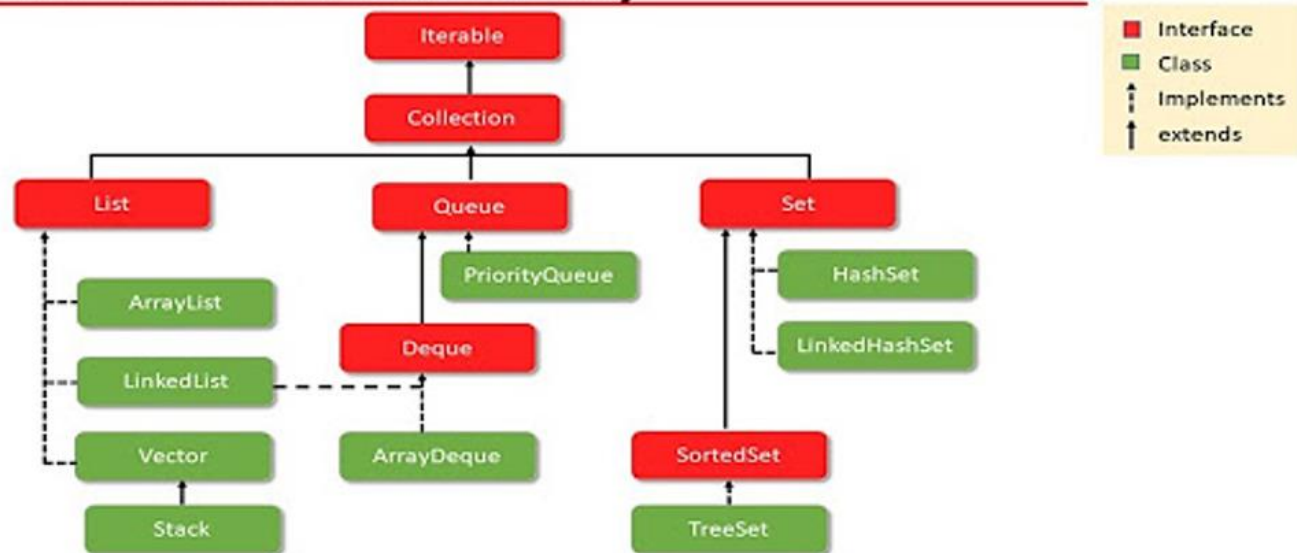
- ✓ Um **HashMap** em **Java** é parte da framework **Collections** e fornece uma maneira de armazenar e gerenciar pares de chave-valor;
- ✓ Ele é uma implementação da interface **Map** e utiliza uma tabela **hash** para armazenar as entradas, o que permite a busca, inserção e remoção de elementos de forma eficiente.



Collections Framework



Java collections hierarchy



HashMap – Características

- ✓ **Chave-Valor:** Cada elemento no **HashMap** consiste em uma chave e um valor associado. Cada chave é única no mapa, enquanto os valores podem ser duplicados;
- ✓ **Performance:** Oferece inserção, busca e exclusão de elementos em tempo constante, assumindo que a função **hash** distribua os elementos uniformemente pela tabela;



HashMap – Características

- ✓ **Ordem:** Não garante nenhuma ordem específica para a iteração dos elementos, pois a ordem é definida pela função de hash das chaves;
- ✓ **Chaves e Valores Nulos:** Permite no máximo uma chave **nula** e múltiplos valores nulos;
- ✓ **Não Sincronizado:** Não é sincronizado (não é thread-safe) por padrão. Isso significa que se múltiplas threads modificarem um **HashMap** simultaneamente, sem a devida sincronização, pode haver um acesso concorrente inseguro.





Como criar um hashMap ?

Como criar um `HashMap` ?

- ✓ Um **HashMap**, em **Java**, é uma estrutura de dados que armazena elementos em pares chave-valor;
- ✓ Cada chave é única, e cada chave mapeia para exatamente um valor.

1. Criando um `HashMap`

```
java
```

```
HashMap<String, Integer> map = new HashMap<>();
```

- `HashMap<String, Integer>`: Declara um `HashMap` onde cada chave é uma `String` e cada valor associado é um `Integer`.
- `new HashMap<>()`: Cria uma nova instância de `HashMap`.




Como adicionar itens em um hashMap ?

Como adicionar itens em um hashMap ?

2. Adicionando Elementos ao `HashMap`

java

 Copy code

```
map.put("chave1", 10);  
map.put("chave2", 20);  
map.put("chave3", 30);
```

- `map.put(key, value)`: Adiciona um par chave-valor ao mapa. Se a chave já existe, seu valor é atualizado com o novo valor fornecido.



Como acessar um item em um hashMap ?

Como acessar um item em um hashMap ?

3. Acessando um Valor

java

Copy code

```
System.out.println("Valor associado à chave2: " + map.get("chave2"));
```

- `map.get(key)`: Retorna o valor associado à chave especificada. Neste caso, imprime o valor associado à `"chave2"`.




Como remover um item de um hashMap ?

Como remover um item de um hashMap ?

4. Removendo um Elemento

java

 Copy code

```
map.remove("chave3");
```

- `map.remove(key)`: Remove o par chave-valor associado à chave especificada do mapa.




Como percorrer os itens de um hashMap ?

Como percorrer os itens de um hashMap ?

5. Iterando sobre o `HashMap` com um `Iterator`

java

 Copy code

```
Iterator<Map.Entry<String, Integer>> iterator = map.entrySet().iterator();
```

- `map.entrySet()`: Retorna um conjunto (`Set`) de entradas do mapa. Cada entrada é um par chave-valor.
- `.iterator()`: Retorna um `Iterator` para esse conjunto de entradas.

Como percorrer os itens de um hashMap ?

```
java Copy code  
  
while (iterator.hasNext()) {  
    Map.Entry<String, Integer> entry = iterator.next();  
    System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());  
}
```

- `iterator.hasNext()`: Verifica se há mais elementos para iterar. Retorna `true` se houver.
- `iterator.next()`: Retorna a próxima entrada (par chave-valor) do iterador.
- `entry.getKey()` e `entry.getValue()`: Obtém a chave e o valor da entrada atual, respectivamente.



Como saber o tamanho de um hashMap ?

Como saber o tamanho de um hashMap ?

```
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        // Cria um novo HashMap
        HashMap<String, Integer> map = new HashMap<>();

        // Adiciona alguns elementos ao map
        map.put("Um", 1);
        map.put("Dois", 2);
        map.put("Três", 3);

        // Obtém o tamanho do map
        int size = map.size();

        // Exibe o tamanho do map
        System.out.println("O tamanho do map é: " + size);
    }
}
```

Exemplo

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class ExemploHashMap {
    public static void main(String[] args) {
        // Criando um HashMap
        HashMap<String, Integer> map = new HashMap<>();

        // Adicionando elementos ao map
        map.put("chave1", 10);
        map.put("chave2", 20);
        map.put("chave3", 30);

        // Acessando um valor
        System.out.println("Valor associado à chave2: " + map.get("chave2"));
    }
}
```


Exemplo

```
// Removendo um elemento
map.remove("chave3");

// Obtendo um Iterator para o conjunto de entradas do map
Iterator<Map.Entry<String, Integer>> iterator = map.entrySet().iterator();

// Iterando sobre o map com um loop 'while'
while (iterator.hasNext()) {
    Map.Entry<String, Integer> entry = iterator.next();
    System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());
}
```

Arquitetura Monolítica – Tarefa 2_02

