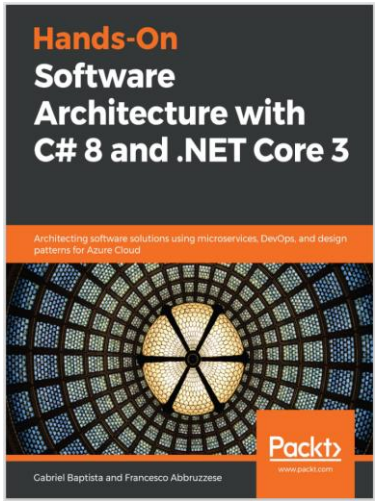
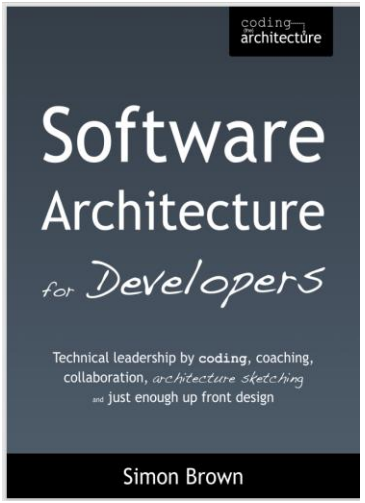
Arquitetura de Software

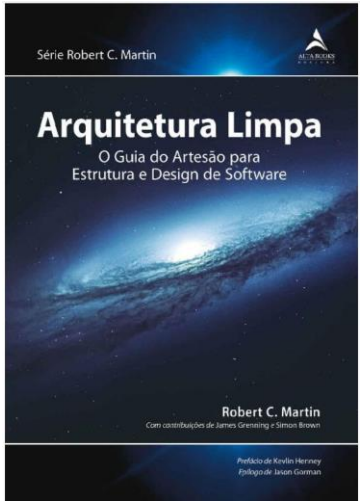
Unidade 4 - Arquitetura em Camadas

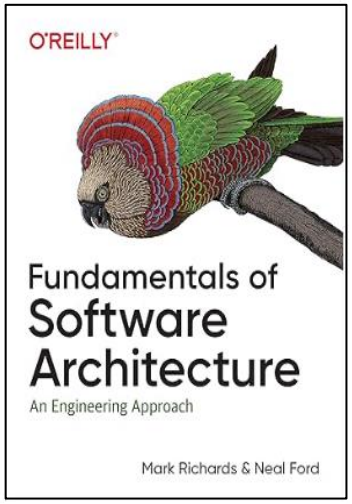
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

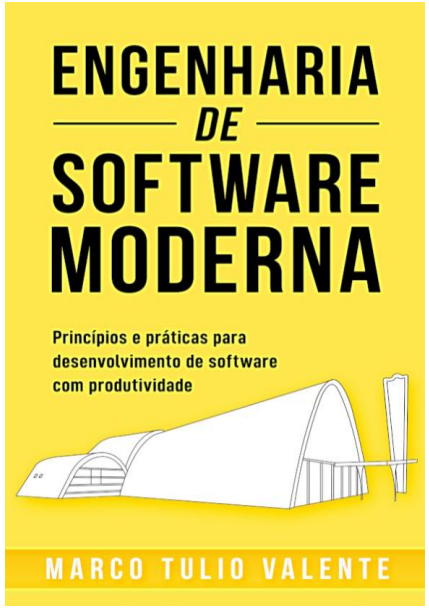
Bibliografia

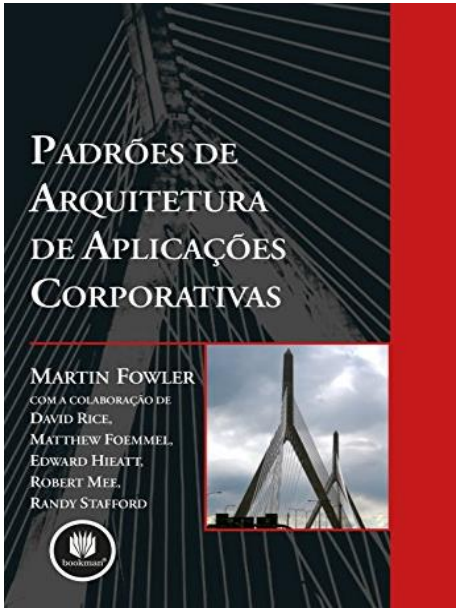












Arquitetura em Camadas

- Um dos **padrões arquiteturais** mais usados, desde que os primeiros sistemas de software de maior porte foram construídos nas **décadas de 60 e 70**;

- Em sistemas que seguem esse padrão, o software é organizado em **módulos** de maior tamanho, chamados de **camadas**.

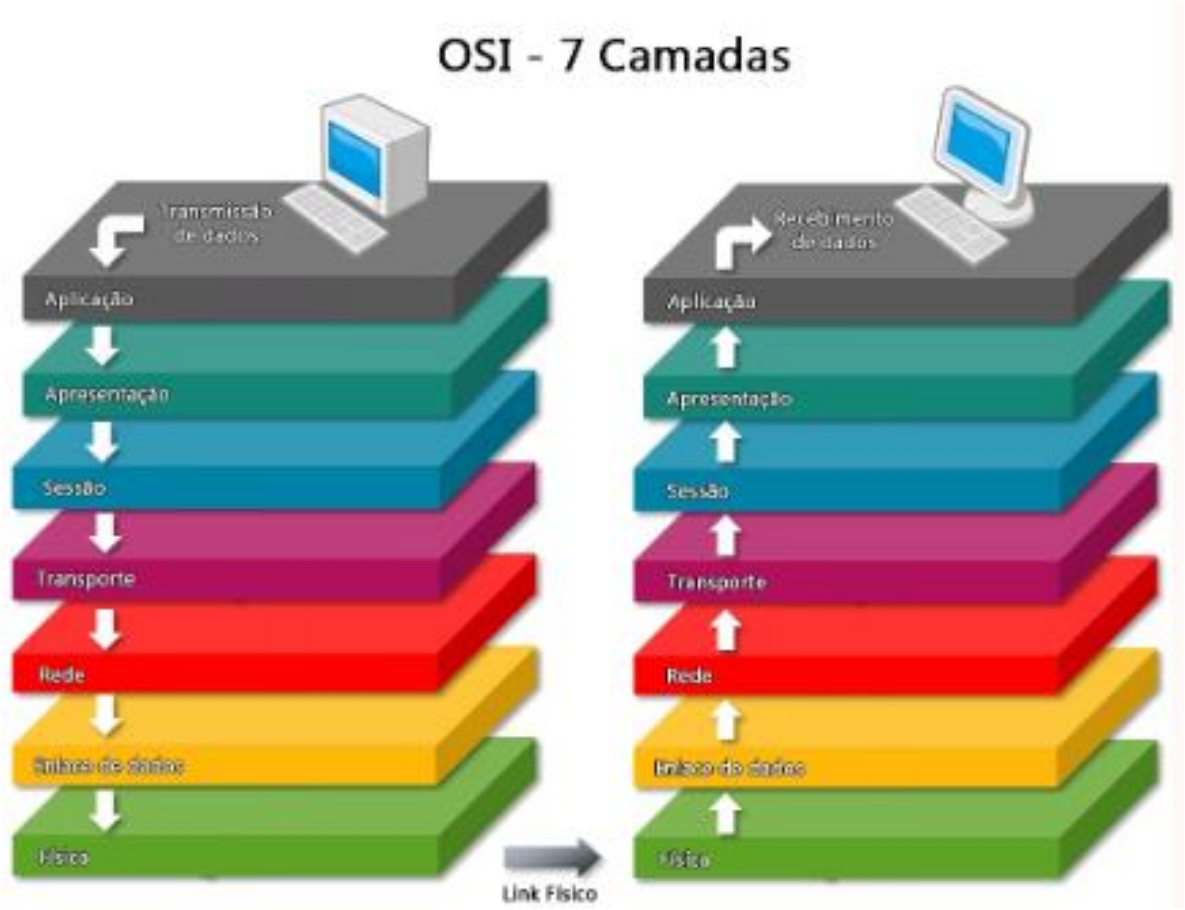
O que seria uma camada ?

Camadas

- Na arquitetura em camadas, uma **camada** é uma coleção **logicamente separada** de funcionalidades que tem um papel específico dentro do software;

- Cada camada tem **responsabilidades únicas** e opera em um nível de abstração diferente.

Arquitetura em Camadas



Arquitetura em Camadas

Arquitetura em Camadas

- O objetivo é separar as preocupações do software de modo que as partes relacionadas a aspectos distintos da aplicação (como **interface de usuário**, **lógica de negócios**, **acesso a dados**, etc.) estejam isoladas umas das outras;

- Isso facilita a **manutenção** e a **escalabilidade**, pois as mudanças em uma **camada** têm impacto **mínimo** ou nenhum sobre as outras.

A modularização pode resultar em uma arquitetura de camadas ?

Revisitando a unidade anterior...

```
package br.uscs;
//-------------------------------------------------
public class DadosQuiz {

      private String[][] capitaisPaises = {

                  { "Itália", "Roma" },
                  { "França", "Paris" },
                  { "Espanha", "Madrid" },
                  { "Alemanha", "Berlim" },
                  { "China", "Pequim" },
                  { "Portugal", "Lisboa" },
                  { "Rússia", "Moscou" },
                  { "Suiça", "Berna" },
                  { "Áustria", "Viena" },
                  { "Egito", "Cairo" } };

//-------------------------------------------------
      public String[][] getCapitaisPaises() {
            return capitaisPaises;
      }
}
//-------------------------------------------------
```

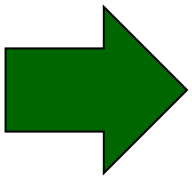
Revisitando a unidade anterior...

```
package br.uscs;
//-------------------------------------------------
public class DadosQuiz {

      private String[][] capitaisPaises = {

                  { "Itália", "Roma" },
                  { "França", "Paris" },
                  { "Espanha", "Madrid" },
                  { "Alemanha", "Berlim" },
                  { "China", "Pequim" },
                  { "Portugal", "Lisboa" },
                  { "Rússia", "Moscou" },
                  { "Suiça", "Berna" },
                  { "Áustria", "Viena" },
                  { "Egito", "Cairo" } };

//-------------------------------------------------
      public String[][] getCapitaisPaises() {
            return capitaisPaises;
      }
}
//-------------------------------------------------
```



Revisitando a unidade anterior...

```
package br.uscs;
//-------------------------------------------------
public class DadosQuiz {

      private String[][] capitaisPaises = {

                  { "Itália", "Roma" },
                  { "França", "Paris" },
                  { "Espanha", "Madrid" },
                  { "Alemanha", "Berlim" },
                  { "China", "Pequim" },
                  { "Portugal", "Lisboa" },
                  { "Rússia", "Moscou" },
                  { "Suiça", "Berna" },
                  { "Áustria", "Viena" },
                  { "Egito", "Cairo" } };

//-------------------------------------------------
      public String[][] getCapitaisPaises() {
            return capitaisPaises;
      }
}
//-------------------------------------------------
```

Módulo de Persistência de Dados

- ✓ Neste design, a classe **DadosQuiz** atua como módulo de **persistência** de dados, responsável apenas por armazenar e fornecer os **dados**.

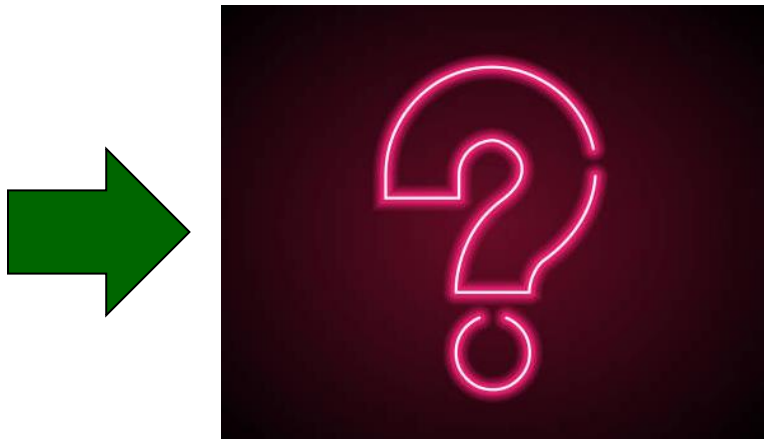
Revisitando a unidade anterior...

```java
package br.uscs;
//------------------------------------------------------------
import java.util.Scanner;
//------------------------------------------------------------
class LogicaQuiz {
	private DadosQuiz dadosQuiz = new DadosQuiz();
	private int score = 0;
//------------------------------------------------------------
	public void processarQuiz() {

		Scanner scanner = new Scanner(System.in);
		String[][] capitaisPaises = dadosQuiz.getCapitaisPaises();

		for (int i = 0; i < capitaisPaises.length; i++) {
			System.out.println("Qual é a capital do país " +
capitaisPaises[i][0] + "?");
			String respostaUsuario = scanner.nextLine();
			verificarResposta(respostaUsuario, capitaisPaises[i][1]);
		}

		scanner.close();
		finalizarQuiz(capitaisPaises.length);
	}
//------------------------------------------------------------------
	private void verificarResposta(String resposta, String respostaCorreta)
{
		if (resposta.trim().equalsIgnoreCase(respostaCorreta)) {
			System.out.println("Correto!");
			score++;
		} else {
			System.out.println("Errado! A resposta correta é: " +
respostaCorreta);
		}
	}
//------------------------------------------------------------------
	private void finalizarQuiz(int totalPerguntas) {
		System.out.println("\nO quiz terminou. Você acertou " + score + "
de " +
				totalPerguntas + " perguntas.");
		int nota = score * 10 / totalPerguntas;
		System.out.println("Sua nota é: " + nota);
		avaliarDesempenho(nota);
	}
//------------------------------------------------------------------
	private void avaliarDesempenho(int nota) {
		if (nota > 8) {
			System.out.println("Ótimo desempenho!");
		} else if (nota < 6) {
			System.out.println("Baixo desempenho!");
		} else {
			System.out.println("Bom desempenho!");
		}
	}
}
//------------------------------------------------------------------
```



Revisitando a unidade anterior...

```java
package br.uscs;
//------------------------------------------------------------
import java.util.Scanner;
//------------------------------------------------------------
class LogicaQuiz {
	private DadosQuiz dadosQuiz = new DadosQuiz();
	private int score = 0;
//------------------------------------------------------------
	public void processarQuiz() {

		Scanner scanner = new Scanner(System.in);
		String[][] capitaisPaises = dadosQuiz.getCapitaisPaises();

		for (int i = 0; i < capitaisPaises.length; i++) {
			System.out.println("Qual é a capital do país " +
capitaisPaises[i][0] + "?");
			String respostaUsuario = scanner.nextLine();
			verificarResposta(respostaUsuario, capitaisPaises[i][1]);
		}

		scanner.close();
		finalizarQuiz(capitaisPaises.length);
	}
//------------------------------------------------------------------
	private void verificarResposta(String resposta, String respostaCorreta)
{
		if (resposta.trim().equalsIgnoreCase(respostaCorreta)) {
			System.out.println("Correto!");
			score++;
		} else {
			System.out.println("Errado! A resposta correta é: " +
respostaCorreta);
		}
	}
//------------------------------------------------------------------
	private void finalizarQuiz(int totalPerguntas) {
		System.out.println("\nO quiz terminou. Você acertou " + score + "
de " +
				totalPerguntas + " perguntas.");
		int nota = score * 10 / totalPerguntas;
		System.out.println("Sua nota é: " + nota);
		avaliarDesempenho(nota);
	}
//------------------------------------------------------------------
	private void avaliarDesempenho(int nota) {
		if (nota > 8) {
			System.out.println("Ótimo desempenho!");
		} else if (nota < 6) {
			System.out.println("Baixo desempenho!");
		} else {
			System.out.println("Bom desempenho!");
		}
	}
}
//------------------------------------------------------------------
```

Módulo – Processar Quiz

- ✓ Lógica de Negócio
- ✓ Apresentação
- ✓ Inicialização

- ✓ A classe **LogicaQuiz** atua como a módulo de processamento do quiz, executando interações com o usuário e lógica de negócios.

A modularização pode resultar em uma arquitetura de camadas ?

A modularização pode resultar em uma arquitetura de camadas ?

- Quando uma aplicação é **modularizada**, seus componentes são divididos em partes distintas que podem ser desenvolvidas, testadas, e mantidas de forma independente;

- Se essa **modularização** seguir uma separação onde um conjunto de módulos se encarrega exclusivamente da **interface com o usuário** e outro conjunto se foca na **lógica de negócios** e **acesso a dados**, então é possível que tal estrutura se assemelhe a uma arquitetura **de duas camadas**.

Reescrevendo a aplicação para 2 camadas

- Para se reestruturar a aplicação em apenas **duas camadas** — uma **camada de Interface com o Usuário** e outra de **Lógica de Negócios e Acesso a Dados** — pode-se combinar a **Lógica** de **Negócios** e o acesso a dados em uma **única camada**;

- Isso **simplifica** a estrutura, mantendo uma clara separação entre a **interação com o usuário** e o **processamento lógico e de dados**.

Reescrevendo a aplicação para 2 camadas

```java
package br.uscs;

public class IniciarQuiz {
    public static void main(String[] args) {
        QuizLogica quizLogica = new QuizLogica();
        QuizInterfaceUsuario quizInterface = new QuizInterfaceUsuario(quizLogica);
        quizInterface.executarQuiz();
    }
}
```

A classe **IniciarQuiz** será responsável por instanciar a interface do usuário e iniciar o processo do quiz.

Reescrevendo a aplicação para 2 camadas

```java
package br.uscs;

public class QuizLogica {
    private String[][] capitaisPaises = {
        {"Itália", "Roma"}, {"França", "Paris"}, {"Espanha", "Madrid"},
        {"Alemanha", "Berlim"}, {"China", "Pequim"}, {"Portugal", "Lisboa"},
        {"Rússia", "Moscou"}, {"Suiça", "Berna"}, {"Áustria", "Viena"},
        {"Egito", "Cairo"}
    };
    private int score = 0;

    public void verificarResposta(String pais, String resposta) {
        for (String[] item : capitaisPaises) {
            if (item[0].equalsIgnoreCase(pais) && item[1].equalsIgnoreCase(resposta)) {
                System.out.println("Correto!");
                score++;
            } else {
                System.out.println("Errado! A resposta correta é: " + item[1]);
            }
            return;
        }
        System.out.println("País não encontrado.");
    }
    public int getScore() {
        return score;
    }
    public int getTotalPerguntas() {
        return capitaisPaises.length;
    }
    public String[][] getCapitaisPaises() {
        return capitaisPaises;
    }
}
```

A classe **QuizLogica** lida com a lógica de negócios e o acesso a dados.

Reescrevendo a aplicação para 2 camadas

```java
package br.uscs;

import java.util.Scanner;

public class QuizInterfaceUsuario {
    private QuizLogica quiz;

    public QuizInterfaceUsuario(QuizLogica quiz) {
        this.quiz = quiz;
    }

    public void executarQuiz() {
        Scanner scanner = new Scanner(System.in);
        String[][] capitaisPaises = quiz.getCapitaisPaises();

        for (int i = 0; i < capitaisPaises.length; i++) {
            System.out.println("Qual é a capital do país " + capitaisPaises[i][0] + "?");
            String respostaUsuario = scanner.nextLine();
            quiz.verificarResposta(capitaisPaises[i][0], respostaUsuario);
        }

        System.out.println("\nO quiz terminou. Você acertou " +
                quiz.getScore() + " de " + quiz.getTotalPerguntas() + " perguntas.");
        scanner.close();
    }
}
```

A classe **QuizInterfaceUsuario** gerencia a interação com o usuário

Reescrevendo a aplicação para 2 camadas

- Agora, a aplicação está bem estruturada em torno de uma arquitetura simplificada de **duas camadas**, com uma classe de entrada dedicada (**IniciarQuiz**) que inicia o quiz;

- Esta organização facilita o entendimento do fluxo do programa e mantém a clareza na separação de responsabilidades.

Design Modular – Tarefa 4_01

```
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes();
var seconds = now.getSeconds();

var ampm = "am";
var colon = '<IMG SRC="images/colon.gif">';

if (hours >= 12) {
    ampm = "pm";
    hours = hours - 12;
}

if (hours == 0) hours = 12;

if (hours < 10) hours = "0" + hours;
else hours = hours + '';

if (minutes < 10) minutes = "0" + minutes;
else minutes = minutes + '';

if (seconds < 10) seconds = "0" + seconds;
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;

Reescreva o código desenvolvido na **Tarefa 2_01** e implemente a aplicação com o emprego de Arquitetura 2 camadas.

Será que seria possível reescrever a aplicação para Arquitetura com 3 camadas ?

Reescrevendo a aplicação para Arquitetura 3 camadas

- Para reescrever a aplicação seguindo uma arquitetura de 3 camadas, precisamos organizar o código em três principais seções: **a camada de apresentação (interface do usuário),** a **camada de lógica de negócios** (**regras do quiz**), e a **camada de acesso a dados** (**informações sobre capitais e países**);

- Esta organização ajuda a separar as **responsabilidades** de cada parte do código, tornando-o mais **modular**, mais fácil de manter e de escalar.

Reescrevendo a aplicação para Arquitetura 3 camadas

```
package br.uscs;

public class IniciarQuiz {

    public static void main(String[] args) {
        QuizLogica quizLogica = new QuizLogica();
        QuizInterfaceUsuario quizInterface = new QuizInterfaceUsuario(quizLogica);
        quizInterface.executarQuiz();
    }
}
```

A classe **IniciarQuiz** é usada para iniciar o aplicativo. Ela cria as instâncias necessárias e inicia a interface do usuário.

Reescrevendo a aplicação para Arquitetura 3 camadas

```
package br.uscs;

public class IniciarQuiz {

    public static void main(String[] args) {
        QuizLogica quizLogica = new QuizLogica();
        QuizInterfaceUsuario quizInterface = new QuizInterfaceUsuario(quizLogica);
        quizInterface.executarQuiz();
    }
}
```

A classe **IniciarQuiz** é usada para iniciar o aplicativo. Ela cria as instâncias necessárias e inicia a interface do usuário.

Reescrevendo a aplicação para Arquitetura 3 camadas

```java
package br.uscs;

import java.util.Scanner;

class QuizInterfaceUsuario {
    private QuizLogica quizLogica;

    public QuizInterfaceUsuario(QuizLogica quizLogica) {
        this.quizLogica = quizLogica;
    }

    public void executarQuiz() {
        Scanner scanner = new Scanner(System.in);
        String[][] capitaisPaises = quizLogica.getCapitaisPaises();

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

        System.out.println("\nO quiz terminou. Você acertou " +
                quizLogica.getScore() + " de " + quizLogica.getTotalPerguntas() + " perguntas.");
        scanner.close();
    }
}
```

A classe **QuizInterfaceUsuario** corresponde à camada de apresentação se concentra na interação com o usuário. Ela chama **a Camada de Lógica de Negócios** para executar o **quiz** e exibe os resultados ao usuário.

Reescrevendo a aplicação para Arquitetura 3 camadas

```
package br.uscs;

class QuizLogica {
    private CapitaisPaisesRepository capitaisPaisesRepo = new CapitaisPaisesRepository();
    private int score = 0;

    public boolean verificarResposta(String pais, String resposta) {
        String respostaCorreta = capitaisPaisesRepo.getCapitalByPais(pais);
        if (respostaCorreta.equalsIgnoreCase(resposta)) {
            score++;
            return true;
        }
        return false;
    }

    public int getScore() {
        return score;
    }

    public int getTotalPerguntas() {
        return capitaisPaisesRepo.getTotalPerguntas();
    }

    public String[][] getCapitaisPaises() {
        return capitaisPaisesRepo.getCapitaisPaises();
    }
}
```

A classe **QuizLogica** corresponde à camada de **Lógica de Negócios** e contém a **lógica central do aplicativo**, como a verificação das respostas e o cálculo da pontuação. Ela chama a **camada de Dados** para checar as respostas do usuário.

Reescrevendo a aplicação para Arquitetura 3 camadas

```
package br.uscs;

class CapitaisPaisesRepository {
    private String[][] capitaisPaises = {
        {"Itália", "Roma"}, {"França", "Paris"}, {"Espanha", "Madrid"},
        {"Alemanha", "Berlim"}, {"China", "Pequim"}, {"Portugal", "Lisboa"}
        {"Rússia", "Moscou"}, {"Suíça", "Berna"}, {"Áustria", "Viena"},
        {"Egito", "Cairo"}
    };

    public String getCapitalByPais(String pais) {
        for (String[] item : capitaisPaises) {
            if (item[0].equalsIgnoreCase(pais)) {
                return item[1];
            }
        }
        return null;
    }

    public int getTotalPerguntas() {
        return capitaisPaises.length;
    }

    public String[][] getCapitaisPaises() {
        return capitaisPaises;
    }
}
```

A classe **CapitaisPaisesRepository** corresponde à **camada de acesso a dados** e é responsável por gerenciar as informações sobre os países e suas capitais.

Design Modular – Tarefa 4_02

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;

Reescreva o código desenvolvido na **Tarefa 2_01** e implemente a aplicação com o emprego de Arquitetura 3 camadas.