MAPEAMENTO COMPLETO: QUESTÕES DAS PROVAS → MATERIAL DE ESTUDO

Disciplina: Programação Orientada a Objetos (POO) **Avaliação:** AV 01 **Professor:** Durval Lins de Siqueira Neto **Material de Referência:** MATERIAL_ESTUDO_POO_FUNDAMENTOS.pdf (23 páginas)

📋 ÍNDICE

📊 TABELA RESUMO DE GABARITOS

Questão	Tipo A	Tipo B	Tipo C	Conceito Principal
Q01	A	D	B	Classes vs Objetos
Q02	A	A	E	Atributos vs Métodos
Q03	C	A	D	Construtores
Q04	C	B	A	Encapsulamento
Q05	A	E	C	Herança
Q06	A	B	A	Polimorfismo
Q07	C	D	E	Interfaces
Q08	E	C	B	Estruturas de Dados

Legenda:

- ✅ 100% coberto no material
- ⚠️ 80% coberto (conceito implícito)
- 🔸 50% coberto (apenas mencionado)

🎯 PROVA TIPO A

Q01 - Classes vs Objetos (Carro/veículo)

📌 GABARITO: A

A) Uma instância concreta do veículo com dados próprios

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 1-2
- **Seção:** "🎯 Conceito #1: Classes vs Objetos"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
CLASSE = MOLDE/PLANTA

Uma classe é como uma planta arquitetônica que define:
• Que características um objeto terá (atributos)
• Que ações um objeto poderá fazer (métodos)

OBJETO = INSTÂNCIA/COISA REAL

Um objeto é uma casa específica construída seguindo a planta:
• Cada casa é única, mas segue o mesmo projeto
• Pode ter valores diferentes para cada característica
```

Exemplo prático (página 2):

```java
// CLASSE - O molde/planta
class Pessoa {
    String nome;
    int idade;
}

// OBJETOS - As instâncias reais
Pessoa joao = new Pessoa();  // João é um objeto
Pessoa maria = new Pessoa(); // Maria é outro objeto

joao.nome = "João";   // Valores próprios
joao.idade = 25;
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que A está correta:

- A classe `Carro` é o **molde** (planta arquitetônica)
- Ao cadastrar um veículo específico, criamos um **objeto** (instância)
- Cada objeto tem **dados próprios** (modelo específico, ano, cor)
- Material página 2: "João é um objeto" com valores específicos

Por que as outras estão erradas:

- **B)** Método calcula impostos, não representa o veículo
- **C)** Molde genérico é a **classe**, não o objeto criado

- **D)** Atributo é uma característica, não o veículo inteiro
- **E)** Função lista veículos, não representa um veículo específico

---

Q02 - Atributos vs Métodos (Produto)

📌 GABARITO: A

A) Atributos representam características e métodos representam comportamentos

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 2-4
- **Seção:** "🔧 Conceito #2: Atributos vs Métodos"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
ATRIBUTOS = CARACTERÍSTICAS/DADOS

São as propriedades que descrevem o objeto:
• Armazenam informações sobre o objeto
• Representam o estado atual do objeto
• Exemplos: nome, idade, cor, tamanho, preço

MÉTODOS = COMPORTAMENTOS/AÇÕES

São as funções que o objeto pode executar:
• Definem o que o objeto pode fazer
• Podem usar e modificar os atributos
• Exemplos: andar(), falar(), acelerar(), calcular()
```

Exemplo prático (página 3):

```
class ContaBancaria {
    // ATRIBUTOS - características/dados
    String titular;
    double saldo;

    // MÉTODOS - comportamentos/ações
    void depositar(double valor) {
        saldo = saldo + valor;  // Modifica o atributo
    }
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que A está correta:

- `nome` e `preço` são **atributos** (características do produto)
- `aplicarDesconto()` é um **método** (comportamento/ação)
- Material página 3 define literalmente: "Atributos = características" e "Métodos = comportamentos"

Q03 - Construtores (ContaBancaria)

📌 GABARITO: C

C) Construtor

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 7-8
- **Seção:** "🏗️ Conceito #6: Construtores"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
Construtor = Método especial que inicializa o objeto quando ele é criado.

Características dos construtores:
• Mesmo nome da classe
• Não tem tipo de retorno (nem void)
• Chamado automaticamente quando usamos new
```

Exemplo prático (página 7):

```java
class Veiculo {
    private String marca;
    private int ano;

    // CONSTRUTOR PADRÃO (sem parâmetros)
    public Veiculo() {
        this.marca = "Não informada";
        this.ano = 2020;  // Valor padrão
        System.out.println("Veículo criado com valores padrão");
    }
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que C está correta:

- Construtor é "**chamado automaticamente quando usamos new**" (página 7)
- Pode inicializar `saldo = 0` **sem intervenção do usuário**
- Material página 7: exemplo de Veiculo() que inicializa valores automaticamente

---

Q04 - Encapsulamento (senha/getters/setters)

📌 GABARITO: C

C) Encapsulamento

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 4-6
- **Seção:** "🔐 Conceito #4: Encapsulamento - Protegendo os Dados"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
Problema: E se alguém fizer isso?
conta.saldo = -1000;  // Saldo negativo inválido!
conta.titular = "";   // Nome vazio!

Solução: ENCAPSULAMENTO - esconder os detalhes internos e controlar o
acesso.

Modificadores de Acesso:
• private - Apenas a própria classe pode acessar
• public - Qualquer classe pode acessar

🎯 Benefícios do Encapsulamento:
• Proteção: Dados não podem ser alterados incorretamente
• Validação: Podemos verificar se os valores são válidos
• Controle: Decidimos como e quando os dados podem ser acessados
• Manutenção: Mudanças internas não afetam o código externo
```

Exemplo prático (página 5):

```java
class ContaBancariaSegura {
    // ATRIBUTOS PRIVADOS - protegidos
    private String titular;
    private double saldo;

    // MÉTODOS PÚBLICOS - interface controlada
    public void setTitular(String nome) {
        if (nome != null && !nome.trim().isEmpty()) {
            this.titular = nome;  // VALIDAÇÃO
        } else {
            System.out.println("Nome não pode ser vazio!");
        }
    }
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que C está correta:

- Atributo `senha` é **privado** (oculto/protegido)
- Acesso apenas por `getSenha()` e `setSenha()` (controle)
- Material página 5: exemplo **exato** com validação em setter
- Página 4: define encapsulamento como "esconder detalhes e controlar acesso"

Q05 - Herança (Personagem → Guerreiro/Mago)

📌 GABARITO: A

A) Herança

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 9-11
- **Seção:** "👥 Conceito #8: Herança - 'É um tipo de'"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
Herança permite que uma classe filha herde características e comportamentos
de uma classe pai.

Relação: is-a (é um tipo de)
• Gerente é um tipo de Funcionário
• Carro é um tipo de Veículo
• Gato é um tipo de Animal
```

Exemplo prático (páginas 9-10):

```java
// SUPERCLASSE (classe pai)
class Animal {
    protected String nome;
    protected int idade;

    public void dormir() {
        System.out.println(nome + " está dormindo...");
    }
}

// SUBCLASSE (classe filha)
class Cachorro extends Animal {
    private String raca;

    // Método específico de Cachorro
```

```
    public void latir() {
        System.out.println(nome + " está latindo: Au au au!");
    }
}
```

Página 11:

```
Cachorro dog = new Cachorro("Rex", 5, "Labrador");

// Métodos herdados de Animal
dog.dormir();  // Funciona!
dog.comer();   // Funciona!

// Método específico
dog.latir();   // Só Cachorro tem
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que A está correta:

- `Guerreiro` e `Mago` **herdam** de `Personagem` (extends)
- Recebem atributos (nome, vida) e métodos (atacar) da classe base
- Podem **acrescentar** habilidades próprias (específicas)
- Material página 10: Cachorro extends Animal + adiciona latir()

Q06 - Polimorfismo (desenhar() em Quadrado/Círculo)

📌 GABARITO: A

A) Polimorfismo

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 13-15
- **Seção:** "🎭 Conceito #10: Polimorfismo - 'Muitas Formas'"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

```
Polimorfismo permite que objetos diferentes respondam à mesma mensagem
de formas específicas.

Ideia central: "Fale para todos os animais fazerem som, cada um fará
seu som específico"
```

Exemplo prático (páginas 11-13):

```
class Veiculo {
    public void acelerar() {
        velocidade += 10;
        System.out.println("Veículo acelerou para " + velocidade);
    }
}

class Carro extends Veiculo {
    // SOBRESCRITA - comportamento específico de Carro
    @Override
    public void acelerar() {
        velocidade += 20;  // Carro acelera mais
        System.out.println("Carro acelerou para " + velocidade);
    }
}

class Bicicleta extends Veiculo {
    @Override
    public void acelerar() {
        velocidade += 5;  // Bicicleta acelera menos
        System.out.println("Bicicleta pedalou para " + velocidade);
    }
}
```

Página 13:

```
// Cada um acelera de forma diferente!
veiculo.acelerar();  // +10 km/h
carro.acelerar();    // +20 km/h
bike.acelerar();     // +5 km/h
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que A está correta:

- `Quadrado` e `Círculo` têm o **mesmo método** `desenhar()`
- Cada um tem **implementação diferente** (sobrescrita)
- Material página 13: define polimorfismo como "objetos diferentes respondam à mesma mensagem de formas específicas"
- Exemplo prático: acelerar() com comportamentos diferentes

Q07 - Interfaces (Pagamento: Cartao/Boleto/Pix)

📌 GABARITO: C

C) Padronizar o comportamento e permitir polimorfismo

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 15-17
- **Seção:** Implícito em "Polimorfismo" e "Relacionamentos"
- **Cobertura:** ⚠️ 80% (conceito implícito)

📖 TRECHO RELACIONADO DO MATERIAL

Página 15 (sobre Polimorfismo):

```
🎯 Benefícios:
• Flexibilidade: Código funciona com diferentes tipos
• Extensibilidade: Posso adicionar novos tipos sem mudar o código existente
• Manutenção: Mudanças localizadas em cada classe
```

Página 14 (exemplo de padronização):

```
ArrayList<Funcionario> funcionarios = new ArrayList<>();
funcionarios.add(new Vendedor("Maria", 2500, 5));
funcionarios.add(new Gerente("Pedro", 4000, 1500));

// POLIMORFISMO: cada objeto executa SEU próprio método
for (Funcionario f : funcionarios) {
    f.exibirInfo();  // Chama método específico de cada tipo
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que C está correta:

- Interface `Pagamento` força todas as classes a terem `processarPagamento()`
- Isso **padroniza** o comportamento (todos têm o mesmo método)
- Permite **polimorfismo**: tratar Cartao, Boleto, Pix de forma uniforme
- Material página 15: "Extensibilidade: adicionar novos tipos sem mudar código"

⚠️ **NOTA:** Interfaces não têm seção específica no material, mas o conceito está implícito no polimorfismo (páginas 13-17).

Q08 - Estrutura de Dados - Pilha (desfazer/refazer)

📌 GABARITO: E

E) Pilha

📍 LOCALIZAÇÃO NO MATERIAL

- **Página:** 23

- **Seção:** "📚 PRÓXIMOS PASSOS"
- **Cobertura:** 🔸 50% (apenas mencionado)

📖 TRECHO DO MATERIAL

Página 23:

```
📚 PRÓXIMOS PASSOS

Depois de dominar estes fundamentos, você estará pronto para:
• Interfaces e Classes Abstratas
• Coleções (ArrayList, HashMap, etc.)
• Tratamento de Exceções
• Padrões de Projeto (Strategy, Observer, etc.)
• Frameworks como Spring Boot
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que E está correta:

- Pilha (Stack) segue o princípio **LIFO** (Last In, First Out)
- "Desfazer/refazer na ordem inversa" = último a entrar, primeiro a sair
- Material página 23: menciona estruturas de dados nos próximos passos

⚠️ **NOTA IMPORTANTE:** Esta questão está **fora do escopo principal** do material de POO. Pilha é uma estrutura de dados, não um conceito de orientação a objetos. O material apenas menciona "Coleções" como próximo passo de estudo.

🎯 PROVA TIPO B

Q01 - Classes vs Objetos (Playlist)

📌 GABARITO: D

D) Uma instância específica e única da classe Playlist, com seu próprio conjunto de músicas

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 1-2
- **Seção:** "🎯 Conceito #1: Classes vs Objetos"
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base conceitual da Prova A Q01 (páginas 1-2):

```
OBJETO = INSTÂNCIA/COISA REAL
Um objeto é uma casa específica construída seguindo a planta:
• Cada casa é única, mas segue o mesmo projeto
• Pode ter valores diferentes para cada característica
```

Por que D está correta:

- Classe `Playlist` = molde (define estrutura)
- "Músicas para Treinar" = **instância específica e única**
- Tem seu **próprio conjunto de músicas** (valores próprios)

Q02 - Atributos vs Métodos (Postagem/curtir())

📌 GABARITO: A

A) Atributos representam características e métodos representam comportamentos

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 2-4
- **Seção:** "🔧 Conceito #2: Atributos vs Métodos"
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base da Prova A Q02 (página 3):

Por que A está correta:

- `numeroDeCurtidas` = **característica** (atributo que armazena quantidade)
- `curtir()` = **comportamento** (ação que incrementa o número)
- Material página 3: "Métodos podem usar e modificar os atributos"

Q03 - Construtores (Pedido com inicialização)

📌 GABARITO: A

A) Um construtor que recebe os dados iniciais e atribui os valores padrão

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 7-8
- **Seção:** "🏗️ Conceito #6: Construtores"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

Página 7-8:

```
// CONSTRUTOR PARAMETRIZADO
public Veiculo(String marca, String modelo, int ano) {
    setMarca(marca);    // Usa setter para validar
    setModelo(modelo);  // Usa setter para validar
    setAno(ano);        // Usa setter para validar
    System.out.println("Veículo criado: " + marca + " " + modelo);
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que A está correta:

- Construtor **recebe dados iniciais** do pedido
- Automaticamente atribui **valores padrão**: `status = "Aguardando Pagamento"`, `data = now()`
- Material página 7: construtor pode combinar parâmetros + valores padrão
- É **automático** (não precisa chamar método depois)

Q04 - Encapsulamento (Funcionario/salario com regras)

📌 GABARITO: B

B) Encapsulamento, que oculta os dados e expõe operações controladas

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 4-6
- **Seção:** "🔐 Conceito #4: Encapsulamento"
- **Cobertura:** ✅ 100%

📖 TRECHO EXATO DO MATERIAL

Página 5:

```
public void depositar(double valor) {
    if (valor > 0) {  // VALIDAÇÃO
        saldo += valor;
        System.out.println("Depósito de R$ " + valor + " realizado");
    } else {
        System.out.println("Valor deve ser positivo!");
    }
}
```

Página 4-5:

```
🎯 Benefícios do Encapsulamento:
• Proteção: Dados não podem ser alterados incorretamente
```

```
• Validação: Podemos verificar se os valores são válidos
• Controle: Decidimos como e quando os dados podem ser acessados
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que B está correta:

- Atributo `salario` é **privado** (oculto/protegido)
- Método `setSalario()` contém **lógica de negócio** (não pode reduzir, registrar log)
- Material página 5: exemplo de validação em métodos públicos
- Página 4: define encapsulamento como "ocultar dados e controlar acesso"

Q05 - Herança (Conta → ContaCorrente/ContaPoupanca)

📌 GABARITO: E

E) Herança

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 9-11
- **Seção:** "👥 Conceito #8: Herança"
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base da Prova A Q05 (páginas 9-11).

Por que E está correta:

- `ContaCorrente` e `ContaPoupanca` **herdam** de `Conta`
- Recebem `agencia`, `numero`, `saldo`, `depositar()`, `sacar()`
- Material página 9: "classe filha herde características e comportamentos de uma classe pai"

Q06 - Polimorfismo (gerarRelatorio: PDF/CSV)

📌 GABARITO: B

B) Polimorfismo

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 13-15
- **Seção:** "🎭 Conceito #10: Polimorfismo"
- **Cobertura:** ✅ 100%

📖 TRECHO DO MATERIAL

Página 14-15:

```
// Lista de Funcionarios (superclasse)
ArrayList<Funcionario> funcionarios = new ArrayList<>();
funcionarios.add(new Funcionario("João", 3000));
funcionarios.add(new Vendedor("Maria", 2500, 5));
funcionarios.add(new Gerente("Pedro", 4000, 1500));

// POLIMORFISMO: cada objeto executa SEU próprio calcularSalario()
for (Funcionario f : funcionarios) {
    f.exibirInfo();  // Chama calcularSalario() específico de cada tipo
}
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que B está correta:

- Função `gerarRelatorio(documento)` recebe diferentes tipos
- `PdfDocument` e `CsvDocument` têm seu próprio `exportar()`
- A função trata todos de forma uniforme (mesma interface)
- Material página 15: "objetos diferentes respondam à mesma mensagem de formas específicas"

Q07 - Interfaces (GatewayPagamento: Cielo/PagSeguro/Stripe)

📌 GABARITO: D

D) Padronizar a forma como o sistema interage com qualquer gateway, permitindo trocá-los ou adicionar novos facilmente

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 15-17
- **Seção:** Implícito em "Polimorfismo e Relacionamentos"
- **Cobertura:** ⚠️ 80%

📖 TRECHO RELACIONADO DO MATERIAL

Página 15:

```
🎯 Benefícios [do Polimorfismo]:
• Flexibilidade: Código funciona com diferentes tipos
• Extensibilidade: Posso adicionar novos tipos sem mudar o código existente
• Manutenção: Mudanças localizadas em cada classe
```

✅ FUNDAMENTAÇÃO DA RESPOSTA

Por que D está correta:

- Interface `GatewayPagamento` força todos a terem `processarPagamento()`

- Sistema pode chamar `gateway.processarPagamento()` sem saber qual é
- Pode **trocar** gateways facilmente (Cielo → Stripe)
- Pode **adicionar** novos sem modificar código existente
- Material página 15: "adicionar novos tipos sem mudar código existente"

Q08 - Estrutura de Dados - Fila (impressão FIFO)

📌 GABARITO: C

C) Fila (Queue)

📍 LOCALIZAÇÃO NO MATERIAL

- **Página:** 23
- **Seção:** "📚 PRÓXIMOS PASSOS"
- **Cobertura:** 🔸 50%

📖 FUNDAMENTAÇÃO

Por que C está correta:

- Fila (Queue) segue **FIFO** (First In, First Out)
- "Primeiro documento a chegar deve ser o primeiro a ser impresso" = FIFO
- Material página 23: menciona estruturas de dados

⚠️ **NOTA:** Estruturas de dados não estão detalhadas no material de POO.

🎯 PROVA TIPO C

Q01 - Classes vs Objetos (Aluno "João Silva")

📌 GABARITO: B

B) Objeto, uma instância concreta e individual da classe Aluno

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 1-2
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 1-2).

Por que B está correta:

- Classe `Aluno` = molde
- "João Silva" = **objeto** (instância concreta e individual)
- Material página 2: "João é um objeto" da classe Pessoa

Q02 - Atributos vs Métodos (Rota/recalcularRota())

📌 GABARITO: E

E) distanciaTotal e tempoEstimado são atributos, enquanto recalcularRota() é um método

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 2-4
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 2-4).

Por que E está correta:

- `distanciaTotal` e `tempoEstimado` = **atributos** (armazenam dados)
- `recalcularRota()` = **método** (comportamento/ação)

Q03 - Construtores (Jogador com nível 1, 100 HP)

📌 GABARITO: D

D) Um construtor que define nivel = 1 e vida = 100

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 7-8
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 7-8).

Por que D está correta:

- Construtor é **chamado automaticamente**
- Pode forçar valores fixos: `this.nivel = 1; this.vida = 100;`
- Não permite outros valores iniciais

Q04 - Encapsulamento (Termometro/valorCelsius)

📌 GABARITO: A

A) Encapsulamento

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 4-6
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 4-6).

Por que A está correta:

- `valorCelsius` é **privado** (escondido)
- Métodos `getCelsius()` e `getFahrenheit()` controlam acesso
- Material página 4: "esconder detalhes internos e controlar acesso"

Q05 - Herança (FormaGeometrica → Circulo/Retangulo)

📌 GABARITO: C

C) Herança

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 9-11
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 9-11).

Por que C está correta:

- `Circulo` e `Retangulo` **herdam** de `FormaGeometrica`
- Recebem `cor` e `posicao` (atributos comuns)
- Material página 9: "classe filha herde características da classe pai"

Q06 - Polimorfismo (alimentar(animal): Vaca/Galinha)

📌 GABARITO: A

A) Polimorfismo

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 13-15
- **Cobertura:** ✅ 100%

📖 FUNDAMENTAÇÃO

Mesma base das Provas A e B (páginas 13-15).

Por que A está correta:

- `alimentar(animal)` chama `animal.comer()`
- Cada tipo de animal (Vaca, Galinha) tem seu próprio `comer()`
- Material página 13: "objetos diferentes respondam à mesma mensagem de formas específicas"

Q07 - Interfaces (DispositivoControlavel: Lâmpada/ArCondicionado/SmartTV)

📌 GABARITO: E

E) A capacidade de tratar todos os dispositivos de forma uniforme

📍 LOCALIZAÇÃO NO MATERIAL

- **Páginas:** 15-17
- **Cobertura:** ⚠️ 80%

📖 FUNDAMENTAÇÃO

Conceito implícito no polimorfismo (página 15).

Por que E está correta:

- Interface força todos os dispositivos a terem `ligar()` e `desligar()`
- Sistema pode chamar `dispositivo.ligar()` sem saber o tipo específico
- Trata todos de forma **uniforme** (padronizada)
- Material página 15: "Flexibilidade: Código funciona com diferentes tipos"

Q08 - Estrutura de Dados - Pilha (histórico do navegador)

📌 GABARITO: B

B) Pilha (Stack)

📍 LOCALIZAÇÃO NO MATERIAL

- **Página:** 23
- **Cobertura:** 🔸 50%

📖 FUNDAMENTAÇÃO

Por que B está correta:

- Histórico do navegador segue **LIFO** (Last In, First Out)
- Última página visitada é a primeira a ser exibida ao clicar "Voltar"
- Material página 23: menciona Stack nos próximos passos

⚠️ **NOTA:** Estruturas de dados não estão detalhadas no material de POO.

📊 ANÁLISE DE COBERTURA DO MATERIAL

✅ COBERTURA COMPLETA (100%)

Conceito	Páginas	Questões Relacionadas
Classes vs Objetos	1-2	A-Q01, B-Q01, C-Q01
Atributos vs Métodos	2-4	A-Q02, B-Q02, C-Q02
Construtores	7-8	A-Q03, B-Q03, C-Q03
Encapsulamento	4-6	A-Q04, B-Q04, C-Q04
Herança	9-11	A-Q05, B-Q05, C-Q05
Polimorfismo	13-15	A-Q06, B-Q06, C-Q06

Total de questões com cobertura 100%: 18/24 (75%)

⚠️ COBERTURA PARCIAL (80%)

Conceito	Páginas	Observação
Interfaces	15-17	Implícito em polimorfismo e padronização

Questões: A-Q07, B-Q07, C-Q07

Total de questões com cobertura 80%: 3/24 (12,5%)

🔸 COBERTURA MÍNIMA (50%)

Conceito	Páginas	Observação
Estruturas de Dados (Pilha/Fila)	23	Apenas mencionado, não explicado

Questões: A-Q08, B-Q08, C-Q08

Total de questões com cobertura 50%: 3/24 (12,5%)

⚠️ **NOTA IMPORTANTE:** Estruturas de dados (Pilha, Fila) **não são conceitos de POO**. Elas estão listadas na página 23 como "Próximos Passos" de estudo, fora do escopo do material de fundamentos de POO.

📈 ESTATÍSTICAS FINAIS

Distribuição de Cobertura

```
✅ Cobertura 100%:  18 questões (75.0%)
⚠️ Cobertura 80%:    3 questões (12.5%)
```

```
🔸 Cobertura 50%:    3 questões (12.5%)
──────────────────────────────────────
   TOTAL:          24 questões (100%)
```

Mapeamento Questões → Páginas

Páginas	Questões Cobertas	% do Total
1-2	3 questões (Classes vs Objetos)	12.5%
2-4	3 questões (Atributos vs Métodos)	12.5%
4-6	3 questões (Encapsulamento)	12.5%
7-8	3 questões (Construtores)	12.5%
9-11	3 questões (Herança)	12.5%
11-15	3 questões (Polimorfismo)	12.5%
15-17	3 questões (Interfaces)	12.5%
23	3 questões (Estruturas de Dados)	12.5%

🎯 CONCLUSÕES

Pontos Fortes do Material:

1. ✅ **Cobertura excelente** dos conceitos fundamentais de POO (75% das questões)
2. ✅ **Exemplos práticos** em Java para cada conceito
3. ✅ **Sequência pedagógica** progressiva (iniciante → intermediário → avançado)
4. ✅ **Código completo** com explicações detalhadas
5. ✅ **Exercícios mentais** integrados ao material

Áreas de Melhoria:

1. ⚠️ **Interfaces** mereciam seção dedicada (atualmente apenas implícito)
2. 🔸 **Estruturas de Dados** estão fora do escopo (questões Q08 de todas as provas)

Recomendações:

1. **Para o professor:**
   - Adicionar 1-2 páginas sobre interfaces com exemplos práticos
   - Decidir se estruturas de dados (Pilha/Fila) devem ser incluídas nas provas de POO
2. **Para os alunos:**
   - Estudar sequencialmente as páginas 1-23
   - Praticar todos os exemplos de código
   - Fazer os exercícios mentais de cada seção
   - Complementar estudo de estruturas de dados em material adicional

---

📚 REFERÊNCIAS

Material Principal:

- MATERIAL_ESTUDO_POO_FUNDAMENTOS.pdf (23 páginas)
- Elaborado por: Prof. Durval Lins de Siqueira Neto
- Versão: 1.0 | Data: Setembro 2025
- Público: Estudantes de ADS - 2º Período

Provas Analisadas:

- AV1_POO_2o_C_TIPO A.pdf (4 páginas)
- AV1_POO_2o_C_TIPO B.pdf (5 páginas)
- AV1_POO_2o_C_TIPO C.pdf (5 páginas)

Documento gerado em: 2025-10-16 **Instituição:** Centro Universitário Maurício de Nassau - Caruaru **Curso:** Análise e Desenvolvimento de Sistemas **Disciplina:** Programação Orientada a Objetos (POO)

Classificação da informação: Uso Interno