

Questões – Os 4 pilares da POO

Questão 1 – Abstração

O conceito de **abstração** em Programação Orientada a Objetos tem como principal objetivo:

- a) Permitir que uma classe herde características de outra classe.
 - b) Esconder todos os atributos de uma classe utilizando **private**.
 - c) Representar apenas as características e comportamentos relevantes de um objeto, ocultando detalhes desnecessários.
 - d) Permitir que um mesmo método tenha várias implementações obrigatoriamente.
 - e) Criar várias instâncias de uma mesma classe.
-

Questão 2 – Abstração

Considere o código:

```
public abstract class Animal {  
    public abstract void emitirSom();  
    public void dormir() {  
        System.out.println("Dormindo...");  
    }  
}
```

Sobre o código apresentado, assinale a alternativa correta:

- a) A classe `Animal` pode ser instanciada diretamente com `new Animal()`.
 - b) O método `emitirSom()` possui uma implementação completa.
 - c) A classe utiliza abstração ao definir um comportamento que deverá ser implementado pelas subclasses.
 - d) O método `dormir()` precisa obrigatoriamente ser implementado pelas subclasses.
 - e) Uma classe abstrata não pode possuir métodos concretos.
-

Questão 3 – Encapsulamento

O **encapsulamento** é utilizado principalmente para:

- a) Permitir que qualquer classe altere diretamente os atributos de um objeto.
 - b) Ocultar os detalhes internos de uma classe e controlar o acesso aos seus dados.
 - c) Criar uma relação de herança entre classes.
 - d) Permitir sobrecarga de métodos.
 - e) Transformar automaticamente uma classe em uma interface.
-

Questão 4 – Encapsulamento

Analise o código:

```
public class Conta {  
    private double saldo;  
  
    public double getSaldo() {  
        return saldo;  
    }  
  
    public void depositar(double valor) {  
        if (valor > 0) {  
            saldo += valor;  
        }  
    }  
}
```

Qual conceito da POO está sendo aplicado?

- a) Herança
 - b) Polimorfismo
 - c) Abstração
 - d) Encapsulamento
 - e) Sobrecarga
-

Questão 5 – Encapsulamento

Por que o atributo `saldo` da classe `Conta` foi declarado como `private`?

- a) Para impedir que a classe seja instanciada.
 - b) Para permitir que qualquer classe altere diretamente o atributo.
 - c) Para proteger o estado interno do objeto e controlar como ele pode ser alterado.
 - d) Para permitir herança múltipla.
 - e) Para transformar o atributo em uma constante.
-

Questão 6 – Herança

Considere:

```
public class Animal {  
    public void comer() {  
        System.out.println("Comendo...");  
    }  
}  
  
public class Cachorro extends Animal {  
    public void latir() {  
        System.out.println("Au au!");  
    }  
}
```

Sobre o código, é correto afirmar que:

- a) `Animal` herda de `Cachorro`.
- b) `Cachorro` herda características e comportamentos de `Animal`.
- c) `Cachorro` não pode possuir métodos próprios.

- d) `extends` indica que `Animal` é uma interface.
 - e) Java não permite herança entre classes.
-

Questão 7 – Herança

Qual palavra-chave é utilizada em Java para estabelecer uma relação de herança entre classes?

- a) `implements`
 - b) `inherits`
 - c) `extends`
 - d) `superclass`
 - e) `instanceof`
-

Questão 8 – Herança

Observe:

```
class Funcionario {  
    protected String nome;  
}  
  
class Gerente extends Funcionario {  
    private double bonus;  
}
```

Nesse exemplo:

- a) `Gerente` é superclasse de `Funcionario`.
 - b) `Funcionario` é subclasse de `Gerente`.
 - c) `Gerente` é uma subclasse de `Funcionario`.
 - d) Não existe herança porque o atributo `nome` é `protected`.
 - e) `Gerente` não possui acesso ao atributo `nome`.
-

Questão 9 – Polimorfismo

O conceito de **polimorfismo** permite:

- a) Que uma classe tenha apenas um objeto.
 - b) Que uma referência de um tipo mais geral possa representar objetos de diferentes subclasses, permitindo comportamentos específicos.
 - c) Que todos os métodos sejam obrigatoriamente `static`.
 - d) Que uma classe herde de várias classes ao mesmo tempo.
 - e) Que atributos `private` sejam acessados diretamente.
-

Questão 10 – Polimorfismo

Considere:

```
class Animal {
    public void emitirSom() {
        System.out.println("Som do animal");
    }
}

class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Au au!");
    }
}

class Gato extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Miau!");
    }
}
```

Agora:

```
Animal animal;
```

```
animal = new Cachorro();
animal.emitirSom();
```

```
animal = new Gato();
animal.emitirSom();
```

Qual será a saída?

a)

```
Som do animal
Som do animal
```

b)

```
Au au!
Miau!
```

c)

```
Cachorro
Gato
```

d)

```
Miau!
Au au!
```

e) O código produzirá um erro de compilação.

Questão 11 – Polimorfismo

No exemplo abaixo:

```
Animal animal = new Cachorro();
```

podemos afirmar que:

- a) Um objeto `Animal` foi convertido em `Cachorro`.
 - b) Uma referência do tipo `Animal` está apontando para um objeto `Cachorro`.
 - c) `Cachorro` deixou de ser uma subclasse de `Animal`.
 - d) A classe `Animal` obrigatoriamente precisa ser abstrata.
 - e) O objeto criado pertence simultaneamente a duas classes independentes.
-

Questão 12 – Os quatro pilares

Associe cada conceito à sua respectiva descrição:

1. Abstração

2. Encapsulamento

3. Herança

4. Polimorfismo

- ☐ Permite que uma classe reutilize características de outra classe.
- ☐ Permite representar diferentes comportamentos por meio de uma mesma interface ou referência.
- ☐ Permite modelar somente os aspectos relevantes de uma entidade, ocultando detalhes desnecessários.
- ☐ Controla o acesso aos dados e detalhes internos de um objeto.

Assinale a sequência correta:

- a) 1 – 2 – 3 – 4
 - b) 3 – 4 – 1 – 2
 - c) 4 – 3 – 2 – 1
 - d) 2 – 1 – 4 – 3
 - e) 3 – 1 – 4 – 2
-

Questão 13 – Análise de código

Analise:

```
public abstract class Forma {  
    private String cor;  
  
    public String getCor() {  
        return cor;  
    }  
  
    public abstract double calcularArea();  
}  
  
public class Retangulo extends Forma {  
    private double largura;  
    private double altura;
```

```
@Override
public double calcularArea() {
    return largura * altura;
}
}
```

Quais pilares da POO podem ser identificados no exemplo?

- a) Apenas encapsulamento.
 - b) Apenas herança.
 - c) Abstração, encapsulamento, herança e polimorfismo.
 - d) Apenas polimorfismo.
 - e) Apenas abstração e encapsulamento.
-

Questão 14 – Questão discursiva

Explique, com suas palavras, o que é **abstração** em Programação Orientada a Objetos e dê um exemplo de como ela pode ser implementada em Java.

Questão 15 – Questão discursiva

Explique o conceito de **encapsulamento** e explique por que normalmente os atributos de uma classe são declarados como `private`.

Questão 16 – Questão discursiva

Explique o conceito de **herança** em Java. Qual é a função da palavra-chave `extends`?

Questão 17 – Questão discursiva

Explique o conceito de **polimorfismo** utilizando o seguinte exemplo:

```
Animal animal;

animal = new Cachorro();
animal.emitirSom();

animal = new Gato();
animal.emitirSom();
```

Questão 18 – Questão discursiva

Explique a diferença entre os quatro pilares da POO:

- Abstração
- Encapsulamento
- Herança

- Polimorfismo

Apresente um exemplo de aplicação de cada conceito em Java.

Gabarito

Questões objetivas

Questão	Resposta	Conceito
1	C	Abstração
2	C	Abstração
3	B	Encapsulamento
4	D	Encapsulamento
5	C	Encapsulamento
6	B	Herança
7	C	Herança
8	C	Herança
9	B	Polimorfismo
10	B	Polimorfismo
11	B	Polimorfismo
12	B	Os quatro pilares
13	C	Os quatro pilares

Questão 14 – Resposta esperada

Abstração consiste em representar os aspectos essenciais de uma entidade, deixando de lado detalhes que não são relevantes para determinado contexto.

Em Java, pode ser implementada, por exemplo, por meio de **classes abstratas e interfaces**:

```
abstract class Animal {  
    public abstract void emitirSom();  
}
```

A classe define o comportamento que é relevante, sem necessariamente fornecer todos os detalhes de sua implementação.

Questão 15 – Resposta esperada

Encapsulamento consiste em proteger os dados e detalhes internos de um objeto, controlando a forma como eles podem ser acessados ou modificados.

Os atributos são frequentemente declarados como **private** para impedir acesso direto externo:

```
class Conta {  
    private double saldo;  
    public void depositar(double valor) {
```

```

        if (valor > 0) {
            saldo += valor;
        }
    }
}

```

Nesse exemplo, a classe controla como o `saldo` pode ser alterado.

Questão 16 – Resposta esperada

Herança permite que uma classe aproveite atributos e métodos de outra classe.

A palavra-chave `extends` estabelece a relação de herança:

```

class Animal {
    public void comer() {
        System.out.println("Comendo");
    }
}

class Cachorro extends Animal {
}

```

Nesse exemplo, `Cachorro` é uma subclasse de `Animal` e pode utilizar o método `comer()` herdado.

Questão 17 – Resposta esperada

O código demonstra **polimorfismo**, pois uma referência do tipo `Animal` pode apontar para diferentes objetos, como `Cachorro` e `Gato`.

```

Animal animal = new Cachorro();
animal.emitirSom();

animal = new Gato();
animal.emitirSom();

```

Quando `emitirSom()` é sobrescrito pelas subclasses, o método executado é determinado pelo **tipo real do objeto**, e não apenas pelo tipo da referência.

Por exemplo:

```

Au au!
Miau!

```

Questão 18 – Resposta esperada

Uma resposta possível:

Pilar	Conceito	Exemplo em Java
Abstração	Representar somente os aspectos relevantes de um objeto.	<code>abstract class Animal</code>

Pilar	Conceito	Exemplo em Java
Encapsulamento	Proteger os dados internos e controlar seu acesso.	atributo <code>private</code> + métodos públicos
Herança	Permitir que uma classe reutilize características de outra.	<code>class Cachorro extends Animal</code>
Polimorfismo	Permitir diferentes comportamentos por meio de uma mesma referência ou contrato.	<code>Animal a = new Cachorro()</code>

Uma forma simples de memorizar:

Abstração → o que o objeto faz

Encapsulamento → como protegemos seus dados

Herança → o que podemos reutilizar

Polimorfismo → diferentes formas de executar um comportamento