Técnico em Desenvolvimento de Sistemas

Disciplina: Desenvolvimento de Sistemas

Atividade – Encapsulamento e Modificadores de Acesso

Exercícios Práticos – Encapsulamento e Modificadores de Acesso em Java

1. Classe Pessoa com encapsulamento

Descrição: Crie uma classe chamada Pessoa com os atributos nome (String), idade (int) e cpf (String).Todos os atributos devem ser private. Implemente os métodos get e set para permitir o acesso e modificação dos atributos.

Dica: Use o modificador private para os atributos. Em seguida, crie métodos getNome(), setNome(String nome) etc., para acessar e modificar os valores.

2. Validação de idade na classe Pessoa

Descrição: Na classe Pessoa, adicione uma verificação no método setIdade(int idade) para impedir que valores negativos sejam atribuídos à idade. Exiba uma mensagem ou ignore a atribuição se o valor for inválido.

Dica: No método setIdade(int idade), use um if para verificar se o valor é negativo. Se for, ignore ou imprima uma mensagem de erro.

3. Classe Produto com encapsulamento

Descrição: Crie uma classe Produto com os atributos nome (String), preco (double) e quantidade (int). Todos os atributos devem ser privados e acessados apenas através de métodos get e set.

Dica: Siga o mesmo padrão do exercício 1: declare atributos private e crie get/set para nome, preco e quantidade.

4. Validação de preço em Produto

Descrição:Na classe Produto, modifique o método setPreco(double preco) para aceitar apenas valores positivos. Rejeite a entrada de preços negativos ou nulos.

Dica: No setPreco(double preco), verifique se o preço é maior que zero. Se for menor ou igual a zero, rejeite a atribuição.

5. Classe Relógio com faixa válida de minutos

Descrição:
Crie uma classe Relógio com um atributo minuto (int).Implemente o método setMinuto(int minuto) de forma que apenas valores entre 0 e 59 sejam aceitos.

Dica: No setMinuto(int minuto), use if (minuto >= 0 && minuto < 60) para aceitar. Senão, rejeite ou imprima erro.

6. Teste de acesso default entre classes do mesmo pacote

Descrição: Crie duas classes no **mesmo pacote**: ClasseA e ClasseB. Na ClasseA, crie um atributo int numero com visibilidade **default** (sem modificador). Na ClasseB, tente acessar esse atributo diretamente.

Dica: O modificador "default" (sem modificador explícito) permite acesso dentro do mesmo pacote. Você poderá acessar o atributo número diretamente.

7. Teste de acesso default entre pacotes diferentes

Descrição:Repita o exercício anterior, mas agora mova ClasseB para **um pacote diferente**. Tente novamente acessar o atributo numero de ClasseA e observe o resultado.

Dica: A partir de outro pacote, o atributo numero com modificador default **não poderá ser acessado**. Isso mostra a importância do controle de acesso.

8. Classe Funcionario com salário mínimo

Descrição: Crie uma classe Funcionario com os atributos privados nome (String), salario (double) e cargo (String).Implemente os métodos get e set para todos os atributos. No método setSalario, rejeite valores abaixo do salário mínimo vigente (ex: 1412.00).

Dica: Crie um método setSalario(double salario) que só aceite valores acima de R$ 1412,00 (por exemplo, o salário mínimo atual). Use if.

9. Classe ContaBancaria com operações

Descrição:Crie uma classe ContaBancaria com os atributos numeroConta (String), saldo (double) e titular (String), todos private.Implemente os métodos depositar(double valor) e sacar(double valor).Não permita acesso direto ao atributo saldo com set.

Dica: Faça o atributo saldo private, e implemente métodos depositar(double valor) e sacar(double valor). Não forneça setSaldo() diretamente.

10. Saque com verificação de saldo

Descrição:Na classe ContaBancaria, no método sacar(double valor), verifique se há saldo suficiente antes de realizar o saque.Não permita que o saldo fique negativo.

Dica: No método sacar, verifique se valor <= saldo. Se não for, não permita a operação e informe o usuário

11. Classe Aluno com média e aprovação

Descrição:Crie a classe Aluno com os atributos nome (String), nota1 e nota2 (double), todos privados.Implemente os métodos calcularMedia() e estaAprovado() (retorna true se média ≥ 6).

Dica: Crie um método calcularMedia() que retorna a média das duas notas. Depois, crie estaAprovado() que retorna true se a média >= 6.

12. Validação de notas do Aluno

Descrição:Nos métodos setNota1 e setNota2, garanta que os valores estejam no intervalo de 0 a 10.Se o valor estiver fora da faixa, exiba uma mensagem e ignore a alteração.

Dica: **Dica**: Nos métodos setNota1(double n) e setNota2(double n), verifique se n está entre 0 e 10 antes de atribuir

13. Protected e herança

Descrição:Crie uma classe ConfiguracaoSistema com um atributo versaoSistema (String) com o modificador protected.Crie uma subclasse e acesse esse atributo diretamente, testando o comportamento de visibilidade herdada.

Dica: **Dica**: Use protected no atributo versaoSistema. Crie uma subclasse SubSistema e acesse diretamente esse atributo, pois heranças podem acessá-lo mesmo fora do pacote.

14. Métodos públicos e privados em Empresa

Descrição:Crie a classe Empresa com dois métodos: exibirInfo() como public e gerarRelatorio() como private.Crie outra classe e tente acessar os dois métodos a partir dela. Verifique o que é permitido.

Dica: Crie dois métodos na classe Empresa. Ao tentar chamar gerarRelatorio() de outra classe, o compilador não permitirá. Mostra o controle de visibilidade.

15. Classe Endereco com validação de CEP

Descrição:Crie uma classe Endereco com os atributos rua, bairro e cep (todos private). Implemente get e set para todos os campos.No setCep(String cep), valide que ele tenha exatamente 8 dígitos numéricos antes de permitir a atribuição.

Dica: No setCep(String cep), use if (cep.matches("\\d{8}")) para garantir que o valor tem exatamente 8 dígitos numéricos.