Lista de Exercícios: Classe Object, @Override e protected

1. Melhorando os Logs do E-commerce "CompraCerta"

O sistema do e-commerce "CompraCerta" está com dificuldades para depurar erros. Quando um problema ocorre ao processar um produto, o sistema de log atual exibe uma mensagem inútil como Produto@1f32e575, o que torna a identificação do item problemático uma tarefa demorada.

- **Sua Tarefa:** Crie uma classe Produto encapsulada com os atributos private String sku, private String nome e private double preco. No main, instancie um produto e imprima o objeto para observar a saída padrão. Em seguida, na classe Produto, sobrescreva (@Override) o método toString() para que ele retorne uma representação textual clara e informativa, como por exemplo: "Produto[SKU: 123AB, Nome: Mouse Gamer, Preço: R$ 150,00]".

2. Produtividade na Geração de toString()

Seu líder técnico na "CompraCerta" gostou da sua melhoria, mas ressaltou que desenvolvedores precisam ser produtivos. Escrever o método toString() manualmente pode ser tedioso e propenso a erros.

- **Sua Tarefa:** Delete o método toString() que você escreveu manualmente no exercício anterior. Agora, use a ferramenta de geração de código da sua IDE (Alt+Insert ou Generate...) para criar o método toString() automaticamente. Analise o código que a IDE gerou.

3. Relatório Base da Clínica "PetSaúde"

A clínica veterinária "PetSaúde" precisa de uma forma padronizada de exibir os dados básicos de qualquer animal em seu sistema.

- **Sua Tarefa:** Utilizando a superclasse Animal que você criou na lista de exercícios anterior (com nome e idade), sobrescreva o método toString() nela. A implementação deve retornar uma String formatada, como por exemplo: "Animal{nome='Rex', idade=5}".

4. Relatório Completo com Reutilização de Lógica

Agora, a clínica precisa de um relatório completo para um Cachorro, que deve incluir tanto os dados gerais de Animal quanto a sua raça específica, mas sem duplicar a lógica de formatação.

- **Sua Tarefa:** Na subclasse Cachorro, sobrescreva também o método toString(). Dentro desta nova implementação, você **deve obrigatoriamente** chamar o toString() da superclasse (super.toString()) para obter a parte do nome e idade, e então concatenar com as informações específicas da raça. O resultado final deve ser algo como: "Cachorro{dadosAnimal=Animal{nome='Rex', idade=5}, raca='Labrador'}".

5. Aplicando o Padrão toString() em Hierarquia

A locadora "Mobilidade Urbana" quer a mesma funcionalidade de relatório para sua frota.

- **Sua Tarefa:** Aplique o mesmo conceito dos exercícios 3 e 4 na hierarquia Veiculo -> Carro. O toString() de Veiculo deve mostrar os dados gerais (marca, modelo, ano), e o toString() de Carro deve chamar super.toString() e adicionar os detalhes específicos do carro (número de portas).

6. Controle de Acesso no "NextGen Bank"

O banco digital "NextGen Bank" está projetando sua hierarquia de contas. A regra de negócio é clara: o saldo de uma Conta é uma informação sensível. Ele não pode ser public, pois qualquer parte do sistema poderia alterá-lo. Mas ele também não pode ser private se as subclasses diretas, como ContaPoupanca, precisarem de acesso direto para realizar operações internas de forma eficiente, como aplicar rendimentos.

- **Sua Tarefa:** Crie uma superclasse Conta com os atributos private String titular e protected double saldo. Crie um construtor e um getter público para o saldo. Em seguida, crie uma subclasse

ContaPoupanca que herda de Conta e possui um método public void renderJuros(double taxa) que calcula o valor do juro e o adiciona diretamente ao saldo herdado (ex: this.saldo += this.saldo * taxa;).

7. Testando a Visibilidade do protected

Ainda no cenário do "NextGen Bank", você precisa provar para sua equipe que o modificador protected funciona como o esperado.

- **Sua Tarefa:** Em uma classe Main, crie uma instância de ContaPoupanca com um saldo inicial. Chame o método renderJuros() e, em seguida, use o getter para imprimir o novo saldo, provando que o método da subclasse conseguiu modificar o saldo. Logo abaixo, tente acessar o saldo diretamente (minhaConta.saldo = 0;) e escreva um comentário explicando o erro de compilação que a IDE irá apontar.

8. O Papel do @Override (Análise de Código)

Seu líder técnico está revisando seu código e pergunta: "Vejo que você usou @Override em todos os seus métodos toString(). Quais são exatamente os dois principais benefícios que essa anotação nos traz?".

- **Sua Tarefa:** Em um comentário no seu código, responda à pergunta do seu líder técnico.

9. O Ancestral Universal (Conceitual)

Um novo estagiário na equipe pergunta: "Eu criei uma classe Cliente sem escrever extends Object, mas meu líder disse que ela ainda assim herda de Object. Como isso é possível?". Ele também pergunta: "Eu sei que toString() vem de Object, mas quais são os outros dois métodos 'famosos' que todo objeto em Java herda?".

- **Sua Tarefa:** Em um comentário, explique ao estagiário por que a herança de Object é implícita e pesquise rapidamente para citar os outros dois métodos fundamentais herdados por todos os objetos em Java.

10. Decisão de Design: Jogo de RPG

Você está projetando a hierarquia de personagens para um jogo. A superclasse Personagem possui o atributo int pontosDeVida. Subclasses como Guerreiro e Mago terão habilidades especiais que precisam ler e modificar diretamente os pontos de vida de forma muito frequente e performática. No entanto, você não quer que a classe principal do jogo (GameEngine) possa alterar a vida de um personagem arbitrariamente (ex: personagem.pontosDeVida = 9999;).

- **Sua Tarefa:** Qual modificador de acesso você escolheria para o atributo pontosDeVida na classe Personagem? public, protected ou private com getters/setters públicos? Justifique sua decisão de design, explicando as vantagens e desvantagens da sua escolha no contexto deste problema.