ATV. 05

1. O paradigma da programação orientada ao objeto está pautado em 4 pilares que a caracterizam:
   1. **Encapsulamento**: trata-se do ato da prática de restringir o acesso direto à dados de um objeto, permitindo a manipulação dos dados através de métodos específicos.
   2. **Abstração**: capacidade de focar em pontos e aspectos essenciais, principais, tornando menos relevante aspectos mais complexos. Isso ocorre através da criação de classes que demonstram conceitos reais.
   3. **Herança**: refere-se ao mecanismo de POO em que é capaz de herdar características de outras classes, tratando-se de receber métodos e atributos. Isso acarretará na criação de uma hierarquia de classes.
   4. **Polimorfismo**: trata-se da capacidade que um objeto tem de assumir diferentes formas, permitindo que ele seja implementado de jeitos variados em classes distintas. Os dois principais tipos de polimorfismo são: sobrecarga e sobrescrita.
   5. No caso do objeto pessoa (instância da classe Pessoa), o método apresentar() da própria classe Pessoa é chamado, exibindo apenas o nome e a idade.
   6. No caso do objeto professor (instância da classe Professor), embora ele seja referenciado como Pessoa, o método apresentar() da classe Professor é chamado graças ao polimorfismo. Primeiro, ele chama o método apresentar() da superclasse Pessoa usando super.apresentar(), que imprime o nome e a idade. Em seguida, ele adiciona a impressão da disciplina.
2. GetSaldo: saldo das contas só pode ser acessado através do método getSaldo(), demonstrando o encapsulamento, pois o atributo saldo está privado e não pode ser acessado diretamente.
3. . . .
4. A segurança de tipos é garantida porque, ao usar generics, o tipo dos objetos na lista é explicitamente declarado como Professor. Assim, o compilador impede que outros tipos de dados sejam adicionados à lista, evitando erros em tempo de execução. Durante a iteração sobre a lista (for (Professor professor : listaDeProfessores)), sabemos que cada item da lista é do tipo Professor, então podemos acessar os métodos dessa classe diretamente sem necessidade de conversão ou verificação de tipo.
   1. **Encapsulamento**: trata-se do ato da prática de restringir o acesso direto à dados de um objeto, permitindo a manipulação dos dados através de métodos específicos.
   2. **Abstração**: capacidade de focar em pontos e aspectos essenciais, principais, tornando menos relevante aspectos mais complexos. Isso ocorre através da criação de classes que demonstram conceitos reais.
5. A principal vantagem do uso de interfaces em Java em vez de herança múltipla é sua simplicidade e flexibilidade, evitando muitos dos problemas associados à herança múltipla. Em Java, uma classe pode herdar de apenas uma classe, uma herança simples, mas pode implementar várias interfaces. Isso permite que a classe integre comportamentos de várias interfaces sem se preocupar com a implementação específica de uma superclasse. Embora à primeira vista a herança múltipla possa parecer útil, ela traz muita complexidade e ambiguidade, como o problema do diamante e a alta dependência de implementações. Por outro lado, o uso de interfaces Java facilita o design, a manutenção, a reutilização e a testabilidade do código, fornecendo uma maneira limpa e flexível de definir contratos.
6. **Polimorfismo**: trata-se da capacidade que um objeto tem de assumir diferentes formas, permitindo que ele seja implementado de jeitos variados em classes distintas. Os dois principais tipos de polimorfismo são: sobrecarga e sobrescrita. Ex: Exemplo: Um método acelerar pode ser implementado de forma diferente para um Carro e uma Moto, embora ambos herdem de Veículo.
7. O uso de exceções em Java é superior às verificações condicionais porque oferece uma maneira mais limpa, organizada e eficiente de lidar com erros, promovendo um código mais legível, modular e fácil de manter. Exceções separam a lógica de negócios do tratamento de erros, facilitam a transferência de problemas para níveis superiores e fornecem uma abordagem sólida para lidar com erros imprevistos sem a necessidade de verificações manuais repetidas ao longo do código.
8. Ao permitir que métodos e coleções sejam parametrizados por tipos em Java, generics garantem a segurança de tipos, eliminando a necessidade de casts explícitos e evitando erros em tempo de execução. A verificação de tipos também ajuda a identificar problemas de tipo antecipadamente, o que resulta em um código mais forte e seguro.
   1. Ex:
9. b
10. a
11. b
12. b
13. c
14. b
15. c
16. c
17. b