Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática

Curso Básico Java – Lista de Exercícios 2 – Partes 5 a 8

1. Herança, polimorfismo e interfaces

1) Crie uma classe para representar uma conta corrente, com métodos para depositar uma quantia, sacar uma quantia e obter o saldo. Para cada saque será debitada também uma taxa de operação equivalente à 0,5% do valor sacado. Crie, em seguida, uma subclasse desta classe anterior para representar uma conta corrente de um cliente especial. Clientes especiais pagam taxas de operação de apenas 0,1% do valor sacado. Faça testes com as duas classes e verifique seus resultados.

2a) Crie uma hierarquia de classes de domínio para uma loja que venda livros, CDs e DVDs. Sobrescreva o método `toString()` para que imprima:

- Para livros: nome, preço e autor;
- Para CDs: nome, preço e número de faixas;
- Para DVDs: nome, preço e duração.

Evite ao máximo repetição de código utilizando a palavra `super` no construtor e no método sobrescrito. Em seguida, crie uma classe `Loja` com o método `main()` que adicione 5 produtos diferentes (a sua escolha) a um vetor e, por fim, imprima o conteúdo do vetor.

2b) Modifique o código do programa anterior, da seguinte forma:

a) Adicione um atributo que represente o código de barras do produto (é um valor obrigatório e, portanto, deve ser pedido no construtor);

b) Sobrescreva o método `equals()` retornando `true` se dois produtos possuem o mesmo código de barras;

c) Na classe `Loja`, implemente um simples procedimento de busca que, dado um produto e um vetor de produtos, indique em que posição do vetor se encontra o produto especificado ou imprima que o mesmo não foi encontrado;

d) No método `Loja.main()`, após a impressão do vetor (feita na questão 2a), escolha um dos 5 produtos e crie duas novas instâncias idênticas a ele: uma com o mesmo código de barras e outra com o código diferente. Efetue a busca deste produto no vetor utilizando as duas instâncias e verifique o resultado.

2c) Ainda modificando o código do programa anterior, faça com que `Produto` implemente a interface `Comparable`, e implemente a comparação por nome. Ao final do método `Loja.main()`, ordene o vetor utilizando o método `java.util.Arrays.sort()` e imprima-o novamente. Depois altere a implementação da comparação para ordenar por preço e verifique o resultado.

3) Crie a seguinte hierarquia de classes:

- Uma interface para representar qualquer forma geométrica, definindo métodos para cálculo do perímetro e cálculo da área da forma;
- Uma classe abstrata para representar quadriláteros. Seu construtor deve receber os tamanhos dos 4 lados e o método de cálculo do perímetro já pode ser implementado;
- Classes para representar retângulos e quadrados. A primeira deve receber o tamanho da base e da altura no construtor, enquanto a segunda deve receber apenas o tamanho do lado;
- Uma classe para representar um círculo. Seu construtor deve receber o tamanho do raio.

No programa principal, pergunte ao usuário quantas formas ele deseja criar. Em seguida, para cada forma, pergunte se deseja criar um quadrado, um retângulos ou um círculo, solicitando os dados necessários para criar a forma. Todas as formas criadas devem ser armazenadas em um vetor. Finalmente, imprima: (a) os dados (lados ou raio); (b) os perímetros; e (c) as áreas de todas as formas. Para (b) e (c), tire vantagem do polimorfismo, enquanto que para (a) utilize `instanceof` e *downcast*.

4) A seguir é dado o código de uma aplicação de agenda, incompleta. Siga os passos abaixo para incrementá-la:

e) Crie uma interface chamada `Contato` com os métodos `getNome()`, `getContato()` e `getTipo()`, todos sem parâmetros e retornando `String`;

f) Coloque a interface criada no pacote `br.ufes.inf.prog3.lista2.exercicio04.dominio`;

g) Analise a classe `AplAgenda` (abaixo). Note que ela encontra-se no pacote `br.ufes.inf.prog3.lista2.exercicio04.aplicacao`, portanto crie-a no local adequado;

h) Crie uma classe chamada `ContatoTelefone` no pacote `agenda.dominio` que implemente a interface `Contato`;

i) Implemente o método `executarAdicionarTelefone()` da classe `AplAgenda`. O método deve pedir o nome e o número do telefone de uma pessoa e adicioná-lo na agenda;

j) Transforme `ContatoTelefone` em classe abstrata e implemente três subclasses dela: `ContatoTelefoneResidencial`, `ContatoTelefoneComercial` e `ContatoTelefoneCelular`;

k) Modifique `AplAgenda` para que aceite os três tipos de contato criados;

l) Implemente outros tipos de contato (fax, e-mail, endereço, etc.) e tire proveito do polimorfismo para adicioná-los à aplicação da agenda.

```
package br.ufes.inf.prog3.lista2.exercicio04.aplicacao;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import br.ufes.inf.prog3.lista2.exercicio04.dominio.Contato;

/** Aplicação da Agenda. */
public final class AplAgenda {
	private static final Scanner scanner = new Scanner(System.in);

	/** Onde são mantidos os contatos. */
	private static List contatos = new ArrayList();

	/** Adiciona um contato. */
	private static void adicionarContato(Contato contato) {
		contatos.add(contato);
	}

	/** Obtém um contato, dado o índice. */
	private static Contato obterContato(int indice) {
		if (indice < contatos.size()) return (Contato)contatos.get(indice);
		else return null;
	}

	/** Imprime todos os contatos e seus índices. */
	private static void imprimirContatos() {
		if (contatos.size() == 0) System.out.println("\tAgenda vazia.");
		else for (int i = 0; i < contatos.size(); i++) {
			Contato contato = (Contato)contatos.get(i);
			System.out.println("\t" + i + ": " + contato.getNome() + " (" +
				contato.getTipo() + ")");
		}
		System.out.println();
```

```
	}

	/** Lê do teclado. */
	private static String lerTeclado() {
		return scanner.nextLine();
	}

	/** Método main. */
	public static void main(String[] args) {
		System.out.println("Bem-vindo à Agenda.\n");
		System.out.println("Digite o comando. '?' para ajuda e 'S' para sair.");
		System.out.print("\n> ");

		// Lê o comando.
		String comando = lerTeclado();

		// Continua pedindo comandos até encontrar o comando S, de sair.
		while (! "S".equalsIgnoreCase(comando)) {

			// Comando ?: ajuda.
			if ("?".equals(comando)) {
				System.out.println(
						"\nCOMANDOS DISPONÍVEIS:\n" +
						" ?: Mostra esta lista de comandos;\n\n" +
						" A: Mostra a agenda;\n" +
						" C: Mostra um contato da agenda;\n" +
						" S: Sai do programa;\n\n" +
						"+T: Adiciona um telefone;\n"
						);
			}

			// Comando A: mostrar agenda.
			else if ("A".equalsIgnoreCase(comando)) {
				System.out.println("\nAGENDA:");
				imprimirContatos();
			}

			// Comando C: mostrar contato.
			else if ("C".equalsIgnoreCase(comando)) {
				executarMostrarContato();
			}

			// Comando +T: adicionar telefone.
			else if ("+T".equalsIgnoreCase(comando)) {
				executarAdicionarTelefone();
			}

			// Lê o próximo comando.
			System.out.print("\n> ");
			comando = lerTeclado();
		}
	}

	/** Comando C: mostrar contato. */
	public static void executarMostrarContato() {
		// Lê o índice.
		System.out.print("\nNúmero: ");
```

```
            String indice = lerTeclado();

            // Verifica se é um número.
            if (indice.matches("[0-9]+")) {
                  // Converte para inteiro.
                  int i = Integer.parseInt(indice);

                  // Verifica se o índice existe.
                  if (i < contatos.size()) {
                        // Imprime o contato.
                        Contato contato = (Contato)contatos.get(i);
                        System.out.println(
                                    "\nNome: " + contato.getNome() +
                                    "\n" + contato.getTipo() + ": " + contato.getContato()
                                    );
                  }

                  // Não existe.
                  else System.out.println("Agenda não contém item de número " + i);
            }

            // Não é número.
            else System.out.println("Não é um número.");
      }

      /** Comando +T: adicionar telefone. */
      public static void executarAdicionarTelefone() { }
}
```

2. Classes internas

NAO FAZER daqui para baixo

5) Siga as instruções abaixo para implementar um exercício de demonstração de classes internas:

- Crie uma interface Lampada com métodos ligar() e desligar(), cujo contrato significa simplesmente imprimir mensagens informativas na tela;
- Crie uma classe FabricaLampada que possui duas classes internas aninhadas que implementam a interface Lampada: Incandescente e Fluorescente;
- Crie ainda um método construir(), que receba um parâmetro indicando o tipo de lâmpada e retorne o objeto Lampada criado;
- Crie uma classe Exercicio01 com o método main(). Pergunte ao usuário qual lâmpada ele quer construir, ligue e desligue a lâmpada.

6) Altere o exercício anterior para que as classes Incandescente e Fluorescente sejam:

- Classes membro;
- Classes locais;
- Classes anônimas.

7) Implemente uma lista encadeada, composta por uma sequência de nós que possuem um elemento (conteúdo) e uma referência ao próximo nó. Implemente a classe que representa o nó e a classe que representa o conteúdo como classes internas da classe lista. A classe elemento deve conter um par de números, de forma que a lista encadeada seja uma lista de pares. Existem quatro tipos de classes internas: aninhadas, membro, locais e anônimas. Use tipos diferentes para implementar o nó e o elemento.