Exercícios de Revisão – Java Básico

(i) Programação básica (estruturada)

1) Faça um programa para calcular o valor das seguintes expressões:

- $S_1 = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \ldots + \frac{99}{50}$
- $S_2 = \frac{2^1}{50} + \frac{2^2}{49} + \frac{2^3}{48} + \ldots + \frac{2^{50}}{1}$ \
- $S_3 = \frac{1}{1} - \frac{2}{4} + \frac{3}{9} - \frac{4}{16} + \frac{5}{25} - \ldots - \frac{10}{100}$

2) Observe a seguinte propriedade que alguns números maiores que 1000 e menores que 9999 possuem:

Número:	abcd
	(ab) + (cd) = (ef)
	$(ef)^2$ = abcd
Exemplo:	3025
	30+25 = 55
	55^2 = 3025

Faça um programa que imprima todos os números que satisfaçam esta propriedade.

3) Faça um programa para calcular o Máximo Divisor Comum entre 2 números passados como argumentos para a execução do programa. Observe as seguintes propriedades do MDC:

- MDC(x, y) = MDC(x − y, y), se x > y;
- MDC(x, y) = MDC(y, x);
- MDC(x, x) = x.

Exemplo: MDC(3,5)=MDC(5,3)=MDC(2,3)=MDC(3,2)=MDC(1,2)=MDC(2,1)=MDC(1,1)=1

(ii) Vetores

4) Considere um vetor de 10 números inteiros positivos maiores que zero e um único número X inteiro, também positivo e maior que zero. Faça um programa para:

(a) ler pelo teclado o vetor;

(b) ler pelo teclado o número X;

(c) encontrar e imprimir o par de posições consecutivas cujas componentes possuem a maior distância entre elas;

(d) verificar se o vetor está em ordem crescente, decrescente ou não ordenado;

(e) dizer quantos números no vetor são maiores que X, menores que X e iguais a X.

5) Verifique se as componentes de um vetor de 10 componentes lidos pelo teclado formam uma progressão aritmética, informando se sim ou se não. Caso forme, imprima o termo inicial e a razão.

6) Faça um programa que leia duas matrizes de reais A e B, com respectivos tamanhos 5 x 4 e 4 x 6, e

imprima a matriz resultante A x B (multiplicação das matrizes), de tamanho 5 x 6.

(iii) Programação orientada a objetos

7) Crie uma classe que representa um ponto no plano cartesiano. Em seguida, crie uma classe que representa um triângulo, reusando a classe anterior por composição. Finalmente, escreva um programa que receba do usuário as coordenadas dos vértices do triângulo e imprima seu perímetro.

8) Crie uma classe que representa um funcionário, registrando seu nome, salário e data de admissão. Em seguida, crie uma classe que represente um departamento de uma empresa, registrando o nome e os funcionários que nele trabalham (para uso de vetores, considere um máximo de 100 funcionários). Por fim, crie uma classe que representa uma empresa, registrando seu nome, CNPJ e departamentos (considere um máximo de 10 departamentos). Faça um programa que:

- Crie uma empresa;
- Adicione a esta empresa alguns departamentos;
- Adicione aos departamentos alguns funcionários;
- Dê aumento de 10% a todos os funcionários de um determinado departamento;
- Transfira um funcionário de um departamento para outro.

É esperado que seu código seja bem encapsulado. Por exemplo, para adicionar um departamento em uma empresa (ou um funcionário a um departamento), não se deve acessar o vetor (ou lista) de departamentos diretamente, mas sim ter um método na classe que representa a empresa para adicionar um departamento.

9) 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.

10) 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, crie quadrados, retângulos e círculos com tamanhos diferentes e armazene num vetor. Em seguida, imprima os dados (lados ou raio), os perímetros e as áreas de todas as formas.

(iv) Classes internas

11) Implemente uma lista encadeada, composta por uma seqüência 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.

12) Crie uma classe X com um atributo privado e um método privado que imprima o valor deste atributo. Construa, então, uma classe Y, interna a X, com um método que modifica o valor do atributo e chama o método privado de X. Crie mais um método em X que instancie um objeto Y e chame seu único método. Teste seu código e analise os resultados. Repita o exercício usando uma classe interna anônima.

(v) Exceções

13) Dado o trecho de código abaixo:

```
int[] vetor = new int[] { 2, 4, 6, 8, 10, 12 };
for (int i = 0; i <= 12; i++) {
      System.out.println(vetor[i]);
}
```

Implemente um programa em Java que execute este trecho. Execute o programa e veja a exceção produzida por acesso a posição fora dos limites do vetor. Trate esta exceção, imprimindo na tela uma mensagem dizendo que o vetor acabou.

14) Dado o trecho de código abaixo:

```
public static void metodo01() {
      Class.forName("ClasseQueNaoExiste");
}
public static void metodo02() {
      java.io.File.createTempFile("pre", "suf");
}
public static void metodo03() {
      Integer.class.newInstance();
}
public static void main(String[] args) {
      metodo01();
      metodo02();
      metodo03();
}
```

Uma classe com estes quatro métodos não compila. Quais passos são necessários para fazê-la compilar? Altere o código para que a classe compile sem erros.

15) Modifique a classe que representa a conta-corrente do exercício 9, lançando uma exceção (criada por você) caso o usuário tente depositar ou sacar um valor negativo. Adicione uma outra exceção (também criada por você) caso o usuário tente sacar um valor maior do que o saldo da conta dele. Trate as exceções no seu programa principal, exibindo mensagens de erro adequadas.

(vi) Manipulação de arquivos

16) Uma loja possui 4 filiais, cada uma com um código de 1 a 4. Um arquivo contendo todas as vendas feitas durante o dia nas quatro filiais é gerado a partir de uma planilha, sendo que cada linha do arquivo contém o número da filial e o valor de uma venda efetuada, separados por vírgula. Ex.:

```
1,189.90
1,45.70
3,29.90
```

```
4,55.00
```

No exemplo acima, a filial 1 fez duas vendas, a primeira totalizando R$ 189,90 e a segunda R$ 45,70. A filial 3 fez uma venda de R$ 29,90 e a 4 também uma de R$ 55,00. Faça um programa que leia este arquivo e informe, ao final, o total e o valor médio das vendas de cada filial.

17) No exercício 8 foram criados objetos representando empresas, departamentos e funcionários. Divida o exercício em 2 partes. Na primeira os objetos devem ser criados e serializados em um arquivo no disco. Na segunda, o arquivo deve ser lido, os objetos restaurados e as operações (aumento de 10%, transferência de departamento) efetuadas.

18) Crie um programa que liste o conteúdo da raiz do seu disco e informe, para cada item, se é um arquivo ou se é um diretório.

19) Escreva um programa que receba um nome de arquivo e uma seqüencia de palavras como argumentos na linha de comando, informe se o arquivo existe ou não, caso não exista, crie-o e, por fim, escreva neste arquivo a seqüencia de palavras passadas como argumentos.

(vii) Utilitários

20) Escreva um programa que informe em que dia da semana cairá uma determinada data. Dica: investigue a API da classe `SimpleDateFormat`.

21) Escreva um programa que, dado um número de dias "x" como parâmetro, imprima: "Daqui a x dias será dia DIA/MÊS/ANO (DIA DA SEMANA)". Imprima o ano com 4 dígitos e o dia da semana por extenso.

22) Altere o programa do exercício anterior para imprimir o dia da semana em uma língua diferente (português, inglês, francês, espanhol, etc.). Dica: você pode tanto alterar o local *default* antes de criar o formatador de datas quanto passar o local no construtor do formatador de datas.

23) Crie um programa que receba como parâmetro um valor em reais e converta para dólares (Estados Unidos) e yenes (Japão). Use um formatador de números apropriado para imprimir o resultado, levando o *locale* em consideração. Se não tiver como obter a cotação do dia, use US$ 1 = R$ 3 e R$ 1 = ¥ 38000.

24) Adicione a seqüência de números 2, 5, 3, 9, 2, 4, 3, 8, 5 a um conjunto (`Set`) e a uma lista (`List`), escolhendo a implementação que desejar. Em seguida imprima o conteúdo de ambas as coleções usando um iterador e analise as diferenças.

25) Escreva uma aplicação de dicionário com três funções: adicionar um termo ao dicionário, procurar um termo no dicionário e listar todos os termos existentes em ordem alfabética. Qual classe você usou para implementar a coleção de palavras? Por quê?

26) Escreva um programa que leia nomes, idades e alturas de várias pessoas e armazene numa lista. Em seguida, imprima o conteúdo desta lista ordenado por nome, depois ordenado por idade e por fim ordenado por altura.

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

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

(b) Coloque a interface criada no pacote `agenda.dominio`;

(c) Analise a classe `AplAgenda` (abaixo). Note que ela encontra-se no pacote `agenda.aplicacao`, portanto crie-a no local adequado;

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

(e) 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;

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

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

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

(i) Faça com que a aplicação grave todos os contatos criados num arquivo ao ser encerrada e que leia este arquivo ao ser iniciada, ou seja, faça com que os contatos sejam persistentes (sugestão: use serialização);

(j) Atualize o código de toda a aplicação para as novas facilidades da versão 5.0 de Java. Por exemplo, faça com que as coleções usem os tipos genéricos e substitua loops for e iteradores pelo *for-each* (`for (o : coleção) { }`).

AplAgenda.java:

```
package agenda.aplicacao;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

import agenda.dominio.Contato;

/** Aplicação da Agenda. */
public final class AplAgenda {
	/** 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() {
		// Cria o leitor.
		BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

		// Tenta ler do teclado. Caso não receba nada, retorna string de fim.
		try { return reader.readLine(); }
		catch (IOException e) { return "S"; }
	}

	/** 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 ("?".equalsIgnoreCase(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 ("+TR".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() { }
}
```