Questão 1 – Encapsulamento e Modificadores de Acesso

Enunciado:
Implemente uma classe ContaBancaria com os atributos privados titular (String) e saldo (double).
Crie métodos públicos depositar(double valor), sacar(double valor) e exibirSaldo().
Garanta que o saldo nunca fique negativo.

Questão 2 – Composição ("tem-um")

Enunciado:
Crie uma classe Livro e uma classe Autor.
A classe Livro deve ter um atributo autor do tipo Autor, representando a relação "tem-um".
Implemente um método exibirDetalhes() que mostre o nome do autor e o título do livro.

Questão 3 – Herança e Polimorfismo

Enunciado:
Crie uma classe Funcionario com o método calcularSalario().
Crie duas subclasses: FuncionarioHorista e FuncionarioAssalariado, sobrescrevendo o método para calcular de formas diferentes.
Demonstre o uso de polimorfismo.

Questão 4 – Interfaces e Múltipla Implementação

Enunciado:
Crie uma interface Imprimivel com o método imprimir().
Crie as classes Relatorio e Contrato que implementem essa interface.
Depois, em um método principal, crie uma lista de objetos Imprimivel e chame imprimir() para cada um.

Questão 5 – Modularização e Pacotes

Enunciado:
Organize as classes Produto, Carrinho e Main em pacotes distintos (model, service, app).
Implemente um sistema simples onde o Carrinho adiciona produtos e exibe o total.

Questão 1 – Tratamento de Exceções: Controle de Entrada Inválida

Enunciado:
Crie um programa que leia um número inteiro informado pelo usuário.
Se o usuário digitar algo que **não seja número**, o programa deve capturar a exceção e exibir a mensagem:

"Entrada inválida. Digite um número inteiro."

Use try-catch e a exceção InputMismatchException.

Questão 2 – Hierarquia de Exceções: Leitura de Arquivo

Enunciado:
Crie um método lerArquivo(String caminho) que tenta abrir e ler um arquivo texto.
Capture as exceções FileNotFoundException e IOException separadamente, exibindo mensagens adequadas.
Se o arquivo for lido com sucesso, mostre seu conteúdo.

Questão 3 – Classes Genéricas: Caixa de Armazenamento

Enunciado:
Implemente uma classe genérica Caixa<T> que armazena um objeto de qualquer tipo.
A classe deve ter métodos guardar(T valor) e abrir().
Demonstre o uso da classe com String, Integer e um objeto de uma classe própria (por exemplo, Produto).

Questão 4 – Stream e Coleções: Processando Nomes

Enunciado:
Dado um List<String> de nomes, crie um programa que:

- filtre apenas nomes que começam com "A";
- transforme todos em maiúsculas;
- ordene alfabeticamente;
- e exiba o resultado na tela.

Use a API de **Streams**.

Questão 5 – Anotações e Reflexão

Enunciado:
Crie uma anotação chamada @InfoAutor com os campos autor() e data().
Depois, crie uma classe Documento que utiliza essa anotação.
Por fim, use **reflexão** para ler os valores da anotação em tempo de execução e exibi-los no console.

```
import java.util.LinkedList;
import java.util.Queue;

public class ExemploFila {
   public static void main(String[] args) {
      // Criação da fila
      Queue<String> fila = new LinkedList<>();

      // Adicionando elementos (enqueue)
      fila.add("João");
      fila.add("Maria");
      fila.add("Pedro");
      fila.add("Ana");

      System.out.println("Fila inicial: " + fila);

      // Removendo elemento (dequeue)
      String atendido = fila.poll(); // remove o primeiro elemento
      System.out.println("Atendido: " + atendido);

      // Exibindo o primeiro da fila (peek)
      System.out.println("Próximo: " + fila.peek());

      // Exibindo a fila atual
      System.out.println("Fila atual: " + fila);

      // Verificando se está vazia
      System.out.println("Fila está vazia? " + fila.isEmpty());
   }
}
```

Método	Descrição	Exemplo
push(E item)	Adiciona um elemento ao topo da pilha.	pilha.push("A");
pop()	Remove e retorna o elemento do topo da pilha.	String topo = pilha.pop();
peek()	Retorna o elemento do topo sem removê-lo.	pilha.peek();
empty()	Retorna true se a pilha estiver vazia.	pilha.empty();
search(Object o)	Retorna a posição (1-based) do elemento, se encontrado.	pilha.search("A");
size()	Retorna a quantidade de elementos.	pilha.size();
contains(Object o)	Verifica se o elemento existe na pilha.	pilha.contains("A");
clear()	Remove todos os elementos da pilha.	pilha.clear();
toString()	Retorna uma representação textual da pilha.	System.out.println(pilha);

```
import java.util.Stack;

public class ExemploPilha {
   public static void main(String[] args) {
      // Criação da pilha
      Stack<String> pilha = new Stack<>();

      // Adicionando elementos (push)
      pilha.push("Livro A");
      pilha.push("Livro B");
      pilha.push("Livro C");

      System.out.println("Pilha inicial: " + pilha);

      // Removendo elemento do topo (pop)
      String removido = pilha.pop();
      System.out.println("Removido do topo: " + removido);

      // Verificando o topo da pilha (peek)
      System.out.println("Topo atual: " + pilha.peek());

      // Verificando se contém um elemento
      System.out.println("Contém 'Livro A'? " + pilha.contains("Livro A"));

      // Exibindo a pilha atual
      System.out.println("Pilha atual: " + pilha);
   }
}
```

Método	Descrição	Exemplo
add(E e)	Adiciona elemento ao final da lista.	lista.add("A");
addFirst(E e)	Adiciona elemento no início.	lista.addFirst("A");
addLast(E e)	Adiciona elemento no final (igual a add).	lista.addLast("B");
get(int index)	Retorna o elemento no índice especificado.	lista.get(0);
set(int index, E e)	Substitui o valor em uma posição.	lista.set(1, "Novo");
remove(int index)	Remove o elemento pelo índice.	lista.remove(0);
remove(Object o)	Remove o primeiro elemento igual ao objeto.	lista.remove("A");
clear()	Remove todos os elementos.	lista.clear();
size()	Retorna o tamanho da lista.	lista.size();
isEmpty()	Verifica se a lista está vazia.	lista.isEmpty();
contains(Object o)	Verifica se contém o elemento.	lista.contains("A");