Instituto Superior Politécnico de Tecnologias e Ciências (ISPTEC)

Departamento de Engenharia e Tecnologias

Mentoria de Programação em Java

Licenciatura em Engenharia Informática

2025/2026

Lista de Exercícios de Orientação a Objetos em Java

Judson Paiva

judson.paiva@isptec.co.ao

Exercício 1 – Classe ContaTelefonica

Descrição do Problema

Modelar uma classe que representa a conta mensal de um serviço de telefonia móvel, com controlo de consumo, limite e cálculo do valor a pagar.

Classe ContaTelefonica

- Atributos (privados):
  - numero (String) – número de telefone.
  - plano (String) – ex.: "Pré-pago", "Pós-pago".
  - limiteDadosMB (int) – limite de dados de internet (em MB).
  - dadosConsumidosMB (int) – total de dados utilizados no mês (em MB).
  - valorMensalBase (double) – valor fixo do plano.
  - valorPorMBExcedente (double) – valor cobrado por MB excedente.
- Métodos essenciais:
  - registarConsumoDados(int mb) – incrementa dadosConsumidosMB.
  - calcularExcedenteMB() – devolve quantos MB foram usados acima do limite (ou zero se não houver excedente).
  - calcularValorAPagar() – devolve o valor total (base + excedente × valorPorMBExcedente).

Conteúdo a ser construído no método principal

No programa principal, criar uma instância de ContaTelefonica com, por exemplo:

- numero: "923456789"
- plano: "Pós-pago"
- limiteDadosMB: 5000
- dadosConsumidosMB: iniciar em 0
- valorMensalBase: 10000.0 (Kz)

- valorPorMBExcedente: 5.0 (Kz/MB)

Simular:

1. Registo de consumo de dados em três momentos distintos, por exemplo:
    - primeiro consumo: 2000 MB
    - segundo consumo: 1500 MB
    - terceiro consumo: 2500 MB
2. Cálculo do excedente de dados ao final:
    - total consumido: 6000 MB
    - limite: 5000 MB → excedente: 1000 MB
3. Cálculo do valor total a pagar, combinando:
    - valor base: 10000.0
    - excedente: 1000 * 5.0 = 5000.0
    - total: 15000.0

Apresentar no ecrã: número da conta, plano, dados consumidos, excedente em MB e valor total a pagar.

Exercício 2 – Classe SensorTemperatura

Descrição do Problema

Modelar um sensor de temperatura que regista leituras e calcula estatísticas simples de monitorização de ambiente.

Classe SensorTemperatura

- Atributos (privados):
    - identificador (String) – ex.: "SALA-01".
    - local (String) – ex.: "Laboratório de Informática".

  - leituraMinima (double) – menor temperatura registada.
  - leituraMaxima (double) – maior temperatura registada.
  - somaLeituras (double) – soma de todas as leituras.
  - quantidadeLeituras (int) – número de leituras já registadas.
- Métodos essenciais:
  - registarLeitura(double valor) – atualiza min, max, soma e contagem.
  - obterMedia() – devolve a temperatura média (soma / quantidade).
  - obterIntervalo() – devolve a diferença entre máxima e mínima.

Conteúdo a ser construído no método principal

Criar um sensor com, por exemplo:

- identificador: "SALA-01"
- local: "Laboratório de Redes"

Registar uma sequência de leituras, por exemplo: 22.5, 23.0, 24.2, 21.8, 23.5.

Após registar todas as leituras, obter:

- temperatura mínima: 21.8
- temperatura máxima: 24.2
- média aproximada das leituras
- intervalo (max − min)

Apresentar no ecrã: identificador, local, min, max, média e intervalo de temperatura.

Exercício 3 – Classe PedidoOnlineSimples

Descrição do Problema

Modelar um pedido de compra online simplificado, focando apenas num item, com cálculo de total e estado do pedido.

Classe PedidoOnline

- Atributos (privados):
  - numeroPedido (String)
  - descricaoProduto (String)
  - quantidade (int)
  - precoUnitario (double)
  - estado (String) – ex.: "NOVO", "PAGO", "ENVIADO".
- Métodos:
  - calcularTotal() – devolve quantidade * precoUnitario.
  - marcarComoPago() – altera o estado para "PAGO".
  - marcarComoEnviado() – altera o estado para "ENVIADO", apenas se já estiver pago.

Conteúdo a ser construído no método principal

Criar um pedido com, por exemplo:

- numeroPedido: "PED-2025-001"
- descricaoProduto: "Headset Bluetooth"
- quantidade: 3
- precoUnitario: 15000.0

Simular:

1. Cálculo do valor total (3 * 15000.0 = 45000.0).
2. Alteração do estado para "PAGO".

3. Posteriormente, alteração do estado para "ENVIADO".

Apresentar o número do pedido, a descrição, o estado final e o valor total.

Exercício 4 – Cliente e Endereço

Descrição do Problema

Modelar um cliente que possui um endereço associado, reforçando o relacionamento "tem um" entre as classes.

Classe Endereco

- Atributos:
  - rua (String)
  - bairro (String)
  - cidade (String)
  - pais (String)

Classe Cliente

- Atributos:
  - nome (String)
  - nif (String)
  - telefone (String)
  - endereco (Endereco)
- Métodos:
  - atualizarTelefone(String novoTelefone) – altera o telefone.
  - atualizarEndereco(Endereco novoEndereco) – troca o endereço associado.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar um endereço inicial, por exemplo:

- rua: "Av. Deolinda Rodrigues, 1000"
- bairro: "Kinaxixe"
- cidade: "Luanda"
- pais: "Angola"

2. Criar um cliente com esse endereço, por exemplo:
   - nome: "João Manuel"
   - nif: "005000123LA044"
   - telefone: "923000111"
   - endereço: o criado no passo anterior.
3. Criar um segundo endereço, representando uma mudança de residência, por exemplo:
   - rua: "Rua Direita do Camama, 250"
   - bairro: "Camama"
   - cidade: "Luanda"
   - pais: "Angola"
4. Atualizar o endereço do cliente para o novo endereço e alterar o telefone.

Ao final, apresentar os dados do cliente, com destaque para o endereço antigo (pode ser descrito em comentários na lógica) e o endereço atualmente associado, evidenciando a relação "Cliente TEM UM Endereco".

Exercício 5 – Livro e Categoria

Descrição do Problema

Modelar a relação entre um livro e a categoria a que pertence (por exemplo, "Tecnologia", "Romance", etc.), utilizando "tem um" para representar a classificação.

Classe Categoria

- Atributos:
  - nome (String) – ex.: "Tecnologia"

- descricao (String) – breve texto sobre o tipo de categoria.

Classe Livro

- Atributos:
  - titulo (String)
  - autor (String)
  - ano (int)
  - categoria (Categoria)
- Métodos:
  - alterarCategoria(Categoria novaCategoria) – associa o livro a uma nova categoria.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar duas categorias, por exemplo:
   - categoria 1:
     - nome: "Tecnologia"
     - descricao: "Livros técnicos e científicos na área de TI e Engenharia"
   - categoria 2:
     - nome: "Gestão"
     - descricao: "Livros de administração, gestão de pessoas e negócios"
2. Criar um livro associado inicialmente à categoria "Tecnologia", por exemplo:
   - titulo: "Introdução a Banco de Dados"
   - autor: "Silberschatz"
   - ano: 2011
   - categoria: categoria 1
3. Em seguida, alterar a categoria do livro para "Gestão", simulando uma reorganização de classificação.

Apresentar o título do livro, o autor e o nome da categoria antes e depois da alteração, destacando a relação de um livro "ter uma" categoria.

Exercício 6 – Fatura e Cliente

Descrição do Problema

Representar uma fatura que "tem um" cliente associado, abordando atributos monetários e de identificação.

Classe Cliente

- Atributos:
  - nome (String)
  - nif (String)
  - telefone (String)

Classe Fatura

- Atributos:
  - numero (String)
  - dataEmissao (String)
  - valor (double)
  - cliente (Cliente)
- Métodos:
  - atualizarValor(double novoValor) – ajusta o valor da fatura, com validação para não aceitar valores negativos.
  - associarCliente(Cliente c) – vincula um cliente à fatura.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar um cliente com dados como:
    - nome: "Empresa XPTO, Lda."
    - nif: "5000009999"
    - telefone: "923999000"
2. Criar uma fatura associando esse cliente, por exemplo:
    - numero: "FT-2025-0001"
    - dataEmissao: "2025-06-01"
    - valor: 250000.0
3. Atualizar o valor da fatura para 275000.0, simulando a inclusão de um serviço adicional.

No final, apresentar o número da fatura, a data, o valor final e o nome do cliente associado, enfatizando que a fatura "tem um" cliente como parte de sua composição.

Exercício 7 – Veículo, Carro e Camiao

Descrição do Problema

Modelar uma hierarquia de herança em que diferentes tipos de veículos compartilham características comuns, mas possuem atributos específicos.

Classe base: Veiculo

- Atributos:
    - matricula (String)
    - marca (String)
    - modelo (String)
- Métodos:
    - descrever() – devolve uma descrição textual básica do veículo.

Subclasse: Carro (É UM Veiculo)

- Atributos adicionais:

  - numeroPortas (int)
  - capacidadePassageiros (int)
- Pode especializar o método descrever() acrescentando informação sobre portas e passageiros.

Subclasse: Camiao (É UM Veiculo)

- Atributos adicionais:
  - capacidadeCargaToneladas (double)
  - numeroEixos (int)
- Pode especializar o método descrever() incluindo dados sobre carga e eixos.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar um objeto Carro com valores como:
   - matricula: "LD-10-20-AA"
   - marca: "Toyota"
   - modelo: "Corolla"
   - numeroPortas: 4
   - capacidadePassageiros: 5
2. Criar um objeto Camiao com valores como:
   - matricula: "LD-30-40-BB"
   - marca: "Mercedes"
   - modelo: "Actros"
   - capacidadeCargaToneladas: 18.0
   - numeroEixos: 3
3. Armazenar os dois objetos numa coleção do tipo Veiculo (por exemplo, um array ou lista conceptual) e percorrer essa coleção chamando o método descrever(), observando como Carro e Camiao são tratados como Veiculo, mas fornecem descrições diferentes.

Apresentar cada descrição no ecrã, destacando o uso da relação "Carro É UM Veiculo" e "Camiao É UM Veiculo".

Exercício 8 – Funcionário, Professor e CoordenadorCurso

Descrição do Problema

Representar uma estrutura académica em que diferentes papéis (professor, coordenador de curso) são tipos específicos de funcionário.

Classe base: Funcionario

- Atributos:
  - nome (String)
  - documento (String)
  - salarioBase (double)
- Método:
  - calcularSalarioMensal() – devolve, por padrão, o salarioBase.

Subclasse: Professor (É UM Funcionario)

- Atributos adicionais:
  - area (String) – ex.: "Engenharia Informática"
  - horasAulaSemana (int)
- Pode especializar calcularSalarioMensal() se necessário (por exemplo, considerando uma gratificação por hora).

Subclasse: CoordenadorCurso (É UM Funcionario)

- Atributos adicionais:
  - cursoResponsavel (String)
  - gratificacaoCoordenacao (double)
- Especializa calcularSalarioMensal() adicionando gratificacaoCoordenacao ao salário base.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar um professor com, por exemplo:
    - nome: "Carlos Alberto"
    - documento: "BI1234567"
    - salarioBase: 300000.0
    - area: "Engenharia Informática"
    - horasAulaSemana: 12
2. Criar um coordenador de curso com:
    - nome: "Helena Sousa"
    - documento: "BI7654321"
    - salarioBase: 350000.0
    - cursoResponsavel: "Engenharia Informática"
    - gratificacaoCoordenacao: 50000.0
3. Guardar ambos numa estrutura do tipo Funcionario e chamar calcularSalarioMensal() para cada um, exibindo o nome e o salário mensal calculado.

Desta forma, evidencia-se a herança ao tratar Professor e CoordenadorCurso como Funcionario, e a especialização ao calcular salários diferentes.

Exercício 9 – Conta, ContaPoupanca e ContaSalario

Descrição do Problema

Modelar uma hierarquia bancária em que diferentes tipos de contas herdam atributos comuns, mas aplicam regras distintas para juros ou movimentações.

Classe base: Conta

- Atributos:
    - numero (String)

  - titular (String)
  - saldo (double)
- Métodos:
  - depositar(double valor) – aumenta o saldo se o valor for positivo.
  - sacar(double valor) – diminui o saldo se houver saldo suficiente.
  - aplicarJuros() – implementação padrão pode ser vazia ou neutra.

Subclasse: ContaPoupanca (É UMA Conta)

- Atributos adicionais:
  - taxaJuros (double) – por exemplo, 0.02 para 2% ao mês.
- Especializa aplicarJuros() para aumentar o saldo de acordo com a taxa.

Subclasse: ContaSalario (É UMA Conta)

- Atributos adicionais:
  - empregador (String)
- Pode restringir a forma de saque (por exemplo, não permitir saldo negativo) ou não aplicar juros.

Conteúdo a ser construído no método principal

No programa principal:

1. Criar uma ContaPoupanca com:
   - numero: "CP-001"
   - titular: "Ana Paula"
   - saldo inicial: 200000.0
   - taxaJuros: 0.02 (2%)
2. Criar uma ContaSalario com:
   - numero: "CS-001"
   - titular: "Rui Manuel"
   - saldo inicial: 100000.0
   - empregador: "Empresa ABC"
3. Efetuar depósitoss e eventuais saques em ambas as contas com valores exemplificativos, por exemplo:

- depositar 50000.0 na poupança
- sacar 30000.0 da conta salário

4. Invocar o método aplicarJuros() apenas para a conta poupança e comparar os saldos antes e depois.

Guardar ambas as contas numa coleção do tipo Conta e percorrê-la para apresentar o número da conta, o titular e o saldo final após as operações, destacando a herança ("ContaPoupanca É UMA Conta", "ContaSalario É UMA Conta") e o comportamento diferenciado do método aplicarJuros().

Exercício 10 – Sistema de Biblioteca Universitária

Descrição do Problema:

Modelar um módulo básico de biblioteca, abordando classes, atributos, encapsulamento, métodos e relacionamentos. O sistema deve permitir o registo de autores, livros, alunos e empréstimos, representando o fluxo de circulação dos livros.

Especificação das classes:

- Classe Autor: atributos nome (String), nacionalidade (String).
- Classe Livro: atributos titulo (String), isbn (String), anoPublicacao (int), exemplaresTotais (int), exemplaresDisponiveis (int); métodos baixarExemplar() e disponibilizarExemplar() com controle de limites.
- Classe Aluno: atributos nome (String), numeroEstudante (String), curso (String); método podeEmprestar() que indica se ainda pode requisitar livros.
- Classe Emprestimo: atributos aluno (Aluno), livro (Livro), dataEmprestimo (String), dataPrevistaDevolucao (String), dataDevolucao (String), multaCalculada (double); método calcularMulta() que determina a multa por atraso.

O que deverá ser construído na classe principal:

a) Criar um registo de autor com valores, por exemplo: nome = "Simon Kendal", nacionalidade = "Britânica".
b) Criar um livro com valores, por exemplo: titulo = "Object Oriented Programming Using Java", isbn = "978-0-12-345678-9", anoPublicacao = 2010, exemplaresTotais = 5, exemplaresDisponiveis = 5.
c) Criar um aluno com dados como: nome = "Maria Silva", numeroEstudante = "2025001", curso = "Engenharia Informática".
d) Registar um empréstimo que associe o aluno ao livro, com datas como: dataEmprestimo = "2025-03-01" e dataPrevistaDevolucao = "2025-03-15".
e) Simular a retirada de um exemplar do livro chamando o método adequado, definir uma dataDevolucao (por exemplo "2025-03-20") e invocar calcularMulta() para atualizar a multaCalculada.
f) Apresentar no ecrã um resumo contendo nome do aluno, título do livro, data de empréstimo, data de devolução e multa calculada, observando a necessidade de uso de métodos de acesso para atributos encapsulados.

Exercício 11 – Sistema de Reservas de Salas de Aula

Descrição do Problema:

Modelar o processo de reserva de salas de aula, incluindo professores, salas e registo das reservas com estado e finalidade.

Especificação das classes:

- Classe Sala: atributos codigo (String), capacidade (int), possuiProjetor (boolean).
- Classe Professor: atributos nome (String), idProfessor (String), departamento (String).

- Classe ReservaSala: atributos sala (Sala), professor (Professor), data (String), horarioInicio (String), horarioFim (String), finalidade (String), situacao (String, por exemplo "PENDENTE", "CONFIRMADA", "CANCELADA"); métodos confirmar() e cancelar() para alterar a situação.

O que deverá ser construído na classe principal:

a) Criar uma sala com dados como: codigo = "A101", capacidade = 40, possuiProjetor = true.
b) Criar um professor com dados como: nome = "João Pereira", idProfessor = "P001", departamento = "Informática".
c) Criar uma reserva ligando o professor à sala, usando valores como: data = "2025-04-10", horarioInicio = "10:00", horarioFim = "12:00", finalidade = "Aula de Programação".
d) Invocar o método que altera a situação da reserva para "CONFIRMADA" e, em seguida, apresentar no ecrã o código da sala, o nome do professor, a data, horários e a situação da reserva.
e) Cancelar a reserva por meio do método apropriado e voltar a exibir a situação, verificando que o estado foi atualizado conforme as regras internas da classe.

Exercício 12 – Sistema de Restaurante Universitário

Descrição do Problema:

Representar um restaurante universitário em que estudantes utilizam saldo para comprar refeições definidas em um cardápio diário.

Especificação das classes:

- Classe Refeicao: atributos nome (String), tipo (String – por exemplo "CAFÉ", "ALMOCO", "JANTAR"), valorSubsidiado (double).

- Classe CardapioDia: atributos data (String), lista de Refeicao; métodos adicionarRefeicao(Refeicao r) e listarRefeicoes().
- Classe Estudante: atributos nome (String), numeroEstudante (String), saldoRefeicao (double); método comprarRefeicao(Refeicao r) que debita o saldo quando há saldo suficiente.

O que deverá ser construído na classe principal:

a) Criar um estudante com saldo inicial adequado, por exemplo: nome = "Carlos Alberto", numeroEstudante = "2025002", saldoRefeicao = 1500.0.
b) Criar duas ou mais refeições, por exemplo: uma de tipo "ALMOCO" com valorSubsidiado = 500.0 e outra de tipo "JANTAR" com valorSubsidiado = 300.0.
c) Criar um cardápio para uma data específica, por exemplo data = "2025-04-11", e adicionar as refeições criadas.
d) Simular a compra de uma refeição pelo estudante, utilizando o método de compra para debitar o saldo conforme o valorSubsidiado da refeição escolhida.
e) Apresentar como saída o saldoRefeicao após a compra e a lista de refeições disponíveis no cardápio do dia, verificando a interação entre as classes e o uso de métodos com parâmetros.

Exercício 13 – Sistema de Transporte Escolar

Descrição do Problema:

Modelar um sistema de transporte escolar que regista autocarros, motoristas e rotas realizadas diariamente.

Especificação das classes:

- Classe Onibus: atributos placa (String), capacidade (int), emOperacao (boolean).

- Classe Motorista: atributos nome (String), numeroCarta (String), validadeCarta (String).
- Classe Rota: atributos descricao (String), horaSaida (String), horaChegadaPrevista (String), onibus (Onibus), motorista (Motorista), emAndamento (boolean); métodos iniciar() e finalizar() para controlar o estado da rota.

O que deverá ser construído na classe principal:

a) Criar um objeto Onibus com valores como: placa = "LD-12-34-AZ", capacidade = 50, emOperacao = false.
b) Criar um motorista com dados como: nome = "Pedro Lima", numeroCarta = "CART12345", validadeCarta = "2027-12-31".
c) Criar uma rota com descrição "Rota ISPTEC - Benfica", horaSaida = "06:30" e horaChegadaPrevista = "07:15", associando o autocarro e o motorista criados.
d) Acionar o método que inicia a rota, atualizando os estados emAndamento e emOperacao, e exibir descrição da rota, nome do motorista e estado atual da rota.
e) Acionar o método que finaliza a rota, atualizar os estados novamente e exibir o estado final, evidenciando a mudança de estado do sistema ao longo do percurso.

Exercício 14 – Sistema de Academia

Descrição do Problema:

Representar uma academia com alunos, exercícios físicos e planos de treino personalizados, evidenciando a relação entre estes elementos.

Especificação das classes:

- Classe AlunoAcademia: atributos nome (String), numeroMatricula (String), pesoAtual (double), altura (double).
- Classe ExercicioFisico: atributos nome (String), grupoMuscular (String), series (int), repeticoes (int).
- Classe PlanoTreino: atributos descricao (String), aluno (AlunoAcademia), lista de ExercicioFisico; métodos adicionarExercicio(ExercicioFisico e) e listarExercicios().

O que deverá ser construído na classe principal:

a) Criar um aluno de academia com valores como: nome = "Ana Paula", numeroMatricula = "AC123", pesoAtual = 68.0, altura = 1.65.
b) Criar dois exercícios físicos, por exemplo: "Supino Reto" para o grupo muscular "Peito" com 3 séries de 12 repetições, e "Agachamento Livre" para "Pernas" com 4 séries de 10 repetições.
c) Criar um plano de treino com descricao como "Treino A - Segunda e Quinta", associando o aluno de academia criado.
d) Adicionar ambos os exercícios ao plano por meio do método apropriado, estabelecendo a relação entre PlanoTreino e ExercicioFisico.
e) Exibir a descrição do plano de treino, o nome do aluno e a lista de exercícios incluídos, verificando a composição de objetos e a utilização de métodos de lista.

Exercício 15 – Conta Bancária com Encapsulamento e Regras

Descrição do Problema:

Modelar uma conta bancária que respeita princípios de encapsulamento e regras de negócio, incluindo limites e validação de operações.

Especificação das classes:

- Classe ContaBancaria: atributos privados numeroConta (String), titular (String), saldo (double), limite (double).
- Métodos principais: getSaldo(), setLimite(double limite) que recusa valores negativos, depositar(double valor) que recusa valores menores ou iguais a zero, sacar(double valor) que só permite saques até saldo + limite.

O que deverá ser construído na classe principal:

a) Criar uma conta bancária com valores como: numeroConta = "0001-01", titular = "João dos Santos", saldo inicial = 0.0, limite = 500.0.
b) Realizar uma operação de depósito com um valor positivo, por exemplo 1000.0, utilizando o método depositar.
c) Realizar um saque com valor dentro do limite permitido, por exemplo 1200.0, verificando o impacto no saldo.
d) Tentar realizar um saque adicional com valor que ultrapasse o saldo disponível somado ao limite, garantindo que a operação seja recusada conforme a lógica interna da classe.
e) Apresentar o saldo após cada operação, observando o efeito do encapsulamento e das validações de negócio nos métodos.

Exercício 16 – Funcionário com Validações

Descrição do Problema:

Definir uma classe de funcionário com validação de dados, destacando o uso de constantes, encapsulamento e métodos de acesso controlado.

Especificação das classes:

- Classe Funcionario: atributos privados nome (String), salarioBase (double), cargo (String), dataAdmissao (String).
- Constante pública SALARIO_MINIMO (double) representando o valor mínimo aceitável.
- Métodos setNome(String), setSalarioBase(double), setCargo(String) com validações que impedem dados inconsistentes, além dos getters correspondentes.

O que deverá ser construído na classe principal:

a) Criar um funcionário com dados válidos, por exemplo: nome = "Carlos Sousa", salarioBase = 60000.0, cargo = "Analista", dataAdmissao = "2022-03-10".
b) Tentar alterar o salário base para um valor inferior ao SALARIO_MINIMO, por exemplo 40000.0, através do método setSalarioBase, avaliando como a classe reage à tentativa inválida.
c) Realizar uma alteração de cargo para um valor válido, como "Engenheiro de Software", utilizando o método setCargo.
d) Exibir o nome, cargo e salárioBase após as tentativas de alteração, evidenciando que os dados finais respeitam as regras de validação implementadas.

Exercício 17 – Produto com Preço Promocional

Descrição do Problema:

Modelar um produto com preço original e preço promocional, garantindo a consistência de valores através de regras de negócio e encapsulamento.

Especificação das classes:

- Classe Produto: atributos privados nome (String), precoOriginal (double), precoPromocional (double).
- Métodos setPrecoOriginal(double) que não aceita valores menores ou iguais a zero, setPrecoPromocional(double) que não permite valores superiores ao precoOriginal, e getPrecoParaVenda() que devolve o preço efetivamente praticado.

O que deverá ser construído na classe principal:

a) Criar um produto com dados como: nome = "Teclado Mecânico", precoOriginal = 30000.0.
b) Definir um preço promocional válido, por exemplo precoPromocional = 25000.0, utilizando o método de configuração adequado.
c) Obter o preço para venda por meio do método getPrecoParaVenda(), verificando se o valor retornado corresponde ao preço promocional definido.
d) Tentar definir um preço promocional inválido (por exemplo 35000.0, superior ao precoOriginal) e analisar o comportamento da classe diante dessa situação.
e) Apresentar o nome do produto, o preço original, o preço promocional atual e o preço efetivo para venda, demonstrando o controle exercido pelas validações internas.

Exercício 18 – Aluno com Média Final Encapsulada

Descrição do Problema:

Modelar um aluno com notas e média final, restringindo o acesso direto à média e centralizando o cálculo em um método específico.

Especificação das classes:

- Classe Aluno: atributos privados nome (String), numeroEstudante (String), nota1 (double), nota2 (double), nota3 (double), mediaFinal (double).
- Métodos para definição de cada nota com validação de intervalo, método calcularMedia() que atualiza mediaFinal, e método getMediaFinal() que permite apenas a leitura da média.

O que deverá ser construído na classe principal:

a) Criar um aluno com dados como: nome = "Mariana Tavares", numeroEstudante = "2025003".
b) Atribuir valores às notas utilizando os métodos apropriados, por exemplo nota1 = 16.0, nota2 = 14.0, nota3 = 18.0, respeitando o intervalo definido.
c) Acionar o método calcularMedia() para atualizar o valor de mediaFinal com base nas notas registadas.
d) Obter o valor de mediaFinal através de getMediaFinal() e apresentar no ecrã o nome do aluno e a média calculada.
e) Verificar que não há forma direta de alterar mediaFinal sem passar pelo método de cálculo, reforçando o conceito de encapsulamento da lógica de negócio.

Exercício 19 – Configuração do Sistema (Static + Final)

Descrição do Problema:

Representar configurações globais de um sistema académico, utilizando membros estáticos e finais, além de controle de ambiente de execução.

Especificação das classes:

- Classe ConfiguracaoSistema: atributos estáticos e finais NOME_SISTEMA (String) e VERSAO (String), e atributo estático privado ambiente (String).

- Métodos estáticos getAmbiente() e setAmbiente(String novoAmbiente) que aceita apenas valores pré-definidos, e construtor privado para impedir instanciação da classe.

O que deverá ser construído na classe principal:

a) Aceder e apresentar o valor de NOME_SISTEMA, por exemplo "Sistema Académico ISPTEC", e de VERSAO, por exemplo "1.0.0".
b) Verificar o valor inicial do ambiente através de getAmbiente(), assumindo um padrão como "DESENVOLVIMENTO".
c) Alterar o ambiente para "PRODUCAO" mediante chamada ao método setAmbiente("PRODUCAO"), respeitando a lista de valores permitidos.
d) Exibir novamente o ambiente atual após a alteração, confirmando a alteração da configuração global.
e) Observar que todas as operações são realizadas sem criar instâncias de ConfiguracaoSistema, reforçando o uso de membros estáticos e a natureza utilitária da classe.

Exercício 20 – Usuário do Sistema com Construtores

Descrição do Problema:

Modelar um utilizador do sistema com diferentes perfis, utilizando construtores sobrecarregados para inicialização flexível e regras de validação.

Especificação das classes:

- Classe Usuario: atributos login (String), senha (String), perfil (String).
- Construtor que recebe login e senha e define perfil padrão como "COMUM".
- Construtor que recebe login, senha e perfil, aceitando por exemplo valores "COMUM" ou "ADMIN".

- Validações para impedir login vazio, senha com comprimento insuficiente e perfil fora dos valores permitidos.

O que deverá ser construído na classe principal:

a) Criar um utilizador com o construtor que utiliza o perfil padrão, por exemplo login = "maria", senha = "senha123".
b) Criar um segundo utilizador com o construtor completo, por exemplo login = "admin", senha = "superSenha", perfil = "ADMIN".
c) Verificar os dados iniciais de cada utilizador, comparando os perfis atribuídos automaticamente e explicitamente.
d) Garantir que senhas muito curtas ou perfis inválidos sejam rejeitados pela lógica interna dos construtores.
e) Apresentar no ecrã o login e o perfil de cada instância, ilustrando o uso de construtores sobrecarregados.

Exercício 21 – Produto com Construtor Obrigatório

Descrição do Problema:

Modelar um produto cujo código, descrição e preço sejam obrigatórios no momento da criação, assegurando consistência desde a instância inicial.

Especificação das classes:

- Classe Produto: atributos codigo (String), descricao (String), preco (double), estoque (int).
- Construtor que recebe codigo, descricao e preco, definindo estoque inicial como 0.
- Regras que impedem preços menores ou iguais a zero e alterações posteriores ao código.

- Método reporEstoque(int quantidade) que aumenta o estoque para quantidades positivas.

O que deverá ser construído na classe principal:

a) Criar um produto com valores como: codigo = "P001", descricao = "Monitor 24"", preco = 120000.0.
b) Repor o estoque com uma quantidade positiva, por exemplo 10 unidades, utilizando o método reporEstoque.
c) Confirmar que o estoque inicial era zero e foi atualizado para o novo valor após a reposição.
d) Evitar qualquer tentativa de alteração do código do produto após a construção, reforçando a imutabilidade desse atributo.
e) Apresentar codigo, descricao, preco e estoque atual, verificando a correta inicialização pelo construtor e a atualização posterior via método específico.

Exercício 22 – Aluguel de Carros com Vários Construtores

Descrição do Problema:

Representar veículos disponíveis para aluguel, utilizando construtores distintos para simplificar ou detalhar a criação de objetos.

Especificação das classes:

- Classe CarroAluguel: atributos placa (String), modelo (String), categoria (String), diaria (double), disponivel (boolean).
- Construtor mínimo que recebe placa e modelo e define categoria padrão, diária padrão e disponivel = true.
- Construtor completo que recebe todos os atributos e permite definir categoria e diaria conforme necessidade, com validações para valores coerentes.

O que deverá ser construído na classe principal:

a) Criar um carro com o construtor mínimo, por exemplo placa = "LD-90-12-AA", modelo = "Toyota Yaris", assumindo categoria padrão "ECONOMICO" e diária padrão configurada internamente.
b) Criar um carro com o construtor completo, por exemplo placa = "LD-11-22-BB", modelo = "BMW Série 3", categoria = "LUXO", diaria = 50000.0, disponivel = true.
c) Comparar as informações de categoria e diária entre os dois veículos criados, evidenciando o impacto dos diferentes construtores.
d) Assegurar que diárias negativas ou nulas sejam recusadas na lógica de construção e configuração da classe.
e) Exibir os dados dos veículos (placa, modelo, categoria, diaria, disponivel) para ilustrar a sobrecarga de construtores.

Exercício 23 – Turma com Capacidade Máxima

Descrição do Problema:

Modelar uma turma de alunos com limite de vagas, controlando a matrícula com base na capacidade máxima definida.

Especificação das classes:

- Classe Turma: atributos codigo (String), capacidadeMaxima (int), lista de Aluno (alunosMatriculados).
- Construtor que recebe codigo e capacidadeMaxima e inicializa a lista de alunos vazia.
- Método matricularAluno(Aluno a) que adiciona o aluno se o número de matriculados ainda for inferior à capacidadeMaxima.

O que deverá ser construído na classe principal:

a) Criar uma turma com codigo = "INF-101" e capacidadeMaxima = 2.
b) Criar três alunos com dados distintos, por exemplo "Joana", "Paulo" e "Miguel", cada um com o seu numeroEstudante.
c) Tentar matricular os três alunos na mesma turma, respeitando a regra de capacidade máxima para aceitar apenas os dois primeiros.
d) Confirmar, através da listagem de alunos da turma, que apenas dois alunos foram efetivamente matriculados.
e) Apresentar os dados da turma e dos alunos matriculados, demonstrando o controle de capacidade e o relacionamento entre Turma e Aluno.

Exercício 24 – ContaCorrente com Construtores

Descrição do Problema:

Definir uma conta corrente com dois modos de criação: com ou sem limite inicial, destacando o efeito do limite nas operações financeiras.

Especificação das classes:

- Classe ContaCorrente: atributos numero (String), titular (String), saldo (double), limite (double).
- Construtor que recebe numero e titular, definindo saldo = 0 e limite = 0.
- Construtor que recebe numero, titular e limiteInicial, definindo saldo = 0 e limite = limiteInicial, desde que não negativo.

O que deverá ser construído na classe principal:

a) Criar uma conta sem limite inicial, por exemplo numero = "CC001", titular = "Rui Manuel".

b) Criar uma conta com limite inicial, por exemplo numero = "CC002", titular = "Helena Costa", limiteInicial = 1000.0.
c) Efetuar depósitos em ambas as contas com valores adequados, como 500.0 em cada uma, utilizando o método de depósito definido na classe.
d) Comparar saldo e limite disponíveis em cada conta, constatando a diferença de comportamento entre as duas formas de construção.
e) Exibir um resumo com numero, titular, saldo e limite de cada conta, reforçando o papel dos construtores na configuração inicial do objeto.

Exercício 25 – Pedido e Itens (Composição)

Descrição do Problema:

Modelar um pedido de compra com itens associados, utilizando composição para representar a dependência entre Pedido e ItemPedido.

Especificação das classes:

- Classe Produto: atributos codigo (String), descricao (String), precoUnitario (double).
- Classe ItemPedido: atributos produto (Produto), quantidade (int), subtotal (double), com subtotal calculado a partir de precoUnitario * quantidade.
- Classe Pedido: atributos numero (String), data (String), lista de ItemPedido, status (String, por exemplo "NOVO", "PAGO", "CANCELADO"); métodos adicionarItem(Produto p, int quantidade) e calcularTotal().

O que deverá ser construído na classe principal:

a) Criar dois produtos, por exemplo: codigo = "P100", descricao = "Caderno", precoUnitario = 500.0; e codigo = "P101", descricao = "Caneta", precoUnitario = 100.0.

b) Criar um pedido com numero = "PED-001" e data = "2025-04-20", iniciando a lista de itens vazia e status apropriado, como "NOVO".
c) Adicionar ao pedido um item com o primeiro produto e quantidade 2, e outro item com o segundo produto e quantidade 5, utilizando o método adicionarItem.
d) Calcular o total do pedido por meio de calcularTotal(), que deve somar os subtotais de todos os itens associados.
e) Apresentar numero do pedido, descrição e quantidade de cada produto, subtotal de cada item e o total final, evidenciando a composição entre Pedido e ItemPedido.

Exercício 26 – Matrícula em Disciplina (Classe de Relacionamento)

Descrição do Problema:

Modelar a matrícula de alunos em disciplinas usando uma classe específica para representar o relacionamento muitos-para-muitos.

Especificação das classes:

- Classe Aluno: atributos nome (String), numeroEstudante (String).
- Classe Disciplina: atributos codigo (String), nome (String), cargaHoraria (int).
- Classe MatriculaDisciplina: atributos aluno (Aluno), disciplina (Disciplina), notaFinal (double), situacao (String, por exemplo "APROVADO", "REPROVADO", "EM_ANDAMENTO"); método atualizarSituacao() com base em regra de nota mínima.

O que deverá ser construído na classe principal:

a) Criar um aluno, por exemplo nome = "Sara", numeroEstudante = "2025007".
b) Criar uma disciplina, por exemplo codigo = "BDII", nome = "Base de Dados II", cargaHoraria = 64.

c) Criar uma matrícula ligando o aluno à disciplina, com notaFinal inicial nula ou padrão e situacao = "EM_ANDAMENTO".
d) Definir a notaFinal para um valor específico, como 14.0, e invocar atualizarSituacao() para recalcular a situação (por exemplo, "APROVADO" para notas acima do limiar).
e) Exibir o nome do aluno, o nome da disciplina, a nota final e a situação resultante, evidenciando o papel da classe de relacionamento MatriculaDisciplina.

Exercício 27 – Reserva de Hotel

Descrição do Problema:
Modelar o processo de reserva de quartos de hotel, relacionando hóspede e quarto e calculando o valor total com base em número de noites.
Especificação das classes:

- Classe Quarto: atributos numero (String), tipo (String, por exemplo "SINGLE", "DUPLO", "SUITE"), precoDiaria (double).
- Classe Hospede: atributos nome (String), documento (String).
- Classe ReservaHotel: atributos codigo (String), hospede (Hospede), quarto (Quarto), dataEntrada (String), dataSaida (String), valorTotal (double); método calcularValorTotal(int numeroDeNoites).

O que deverá ser construído na classe principal:

a) Criar um hóspede com dados como: nome = "Luís António", documento = "AB1234567".
b) Criar um quarto com atributos como: numero = "101", tipo = "SINGLE", precoDiaria = 15000.0.
c) Criar uma reserva com codigo = "RES-001", associando o hóspede e o quarto, com dataEntrada = "2025-04-21" e dataSaida = "2025-04-25".
d) Invocar o método calcularValorTotal() com numeroDeNoites = 4, utilizando o precoDiaria do quarto para determinar o valorTotal.

e) Exibir o nome do hóspede, o número e tipo do quarto e o valor total da reserva, reforçando o relacionamento entre ReservaHotel, Hospede e Quarto.

Exercício 28 – Agenda com Compromissos

Descrição do Problema:

Representar uma agenda digital com compromissos diários, mostrando a associação entre uma agenda e múltiplos compromissos.

Especificação das classes:

- Classe Compromisso: atributos data (String), hora (String), descricao (String), local (String).
- Classe Agenda: atributos dono (String), lista de Compromisso; métodos adicionarCompromisso(Compromisso c) e listarCompromissos().

O que deverá ser construído na classe principal:

a) Criar uma agenda com dono = "Diretor Académico".
b) Criar dois compromissos para a mesma data, por exemplo: data = "2025-04-22", horários "09:00" e "11:00", com descrições como "Reunião com coordenação" e "Entrevista de professor", em locais distintos.
c) Adicionar os compromissos à agenda através do método adicionarCompromisso.
d) Utilizar o método listarCompromissos para percorrer e apresentar todos os compromissos registados, mostrando data, hora, descrição e local.
e) Observar a relação entre a Agenda e os Compromisso associados, evidenciando uma estrutura de agregação.

Exercício 29 – Empresa e Funcionários

Descrição do Problema:

Modelar uma empresa e os seus funcionários, incluindo operações de admissão, demissão e cálculo da folha salarial.

Especificação das classes:

- Classe Funcionario: atributos nome (String), cargo (String), salario (double).
- Classe Empresa: atributos nome (String), nif (String), lista de Funcionario; métodos admitir(Funcionario f), demitir(Funcionario f) e calcularFolhaSalarial() que soma os salários dos funcionários activos.

O que deverá ser construído na classe principal:

a) Criar uma empresa com nome = "Tech Angola" e nif = "5000000000".
b) Criar dois funcionários com cargos e salários distintos, por exemplo: "Ana" como "Desenvolvedora" com salario = 250000.0, e "Paulo" como "Analista de Dados" com salario = 300000.0.
c) Admitir ambos os funcionários na empresa através do método admitir.
d) Calcular o valor total da folha salarial utilizando o método calcularFolhaSalarial().
e) Apresentar o nome da empresa, a lista de funcionários (com nome, cargo e salário) e o total da folha, demonstrando a agregação entre Empresa e Funcionario.

Exercício 30 – Hierarquia de Funcionários

Descrição do Problema:

Definir uma hierarquia de funcionários utilizando herança, permitindo o cálculo polimórfico de salários mensais.

Especificação das classes:

- Classe Funcionario: atributos nome (String), documento (String), salarioBase (double); método calcularSalarioMensal() que pode devolver salarioBase.
- Classe FuncionarioHorista (herda de Funcionario): atributos horasTrabalhadas (int), valorHora (double); sobrescreve calcularSalarioMensal() para usar horasTrabalhadas * valorHora.
- Classe FuncionarioAssalariado (herda de Funcionario): utiliza salarioBase directamente como salário mensal.

O que deverá ser construído na classe principal:

a) Criar um FuncionarioHorista com dados como: nome = "Carlos", documento = "123", salarioBase = 0.0, horasTrabalhadas = 160, valorHora = 2000.0 / 160.
b) Criar um FuncionarioAssalariado com dados como: nome = "Helena", documento = "456", salarioBase = 300000.0.
c) Reunir ambas as instâncias numa estrutura comum do tipo Funcionario (por exemplo, um array ou lista).
d) Percorrer a estrutura, invocando calcularSalarioMensal() para cada objecto, sem testar o tipo concreto, ilustrando o polimorfismo.
e) Apresentar o nome de cada funcionário e o salário calculado, evidenciando o comportamento diferente das subclasses sob a mesma interface de método.

Exercício 31 – Formas Geométricas (Classe Abstrata + Polimorfismo)

Descrição do Problema:

Modelar formas geométricas utilizando uma classe abstrata e subclasses concretas, com cálculo de áreas de forma polimórfica.

Especificação das classes:

- Classe abstrata Forma: declara método abstrato double calcularArea().
- Classe Circulo (herda de Forma): atributo raio (double); implementa calcularArea() com a fórmula adequada.
- Classe Retangulo (herda de Forma): atributos largura (double), altura (double); implementa calcularArea() como largura * altura.

O que deverá ser construído na classe principal:

a) Criar um Circulo com raio = 5.0 e um Retangulo com largura = 4.0 e altura = 6.0.
b) Guardar ambas as instâncias numa coleção do tipo Forma, reforçando o uso do tipo abstrato como referência.
c) Percorrer a coleção e invocar calcularArea() para cada forma, sem verificar o tipo concreto, utilizando apenas a referência do tipo Forma.
d) Para cada elemento, apresentar um identificador textual (por exemplo "Círculo" ou "Retângulo") e o valor da área calculada.
e) Observar que o método calcularArea() tem implementações diferentes, mas é chamado de forma uniforme através do polimorfismo.

Exercício 32 – Sistema de Pagamento (Interface)

Descrição do Problema:

Representar um sistema de pagamentos em que diferentes meios de pagamento implementam a mesma interface, permitindo processamento uniforme.

Especificação das classes:

- Interface Pagavel: declara método void processarPagamento(double valor).
- Classe Boleto: atributos codigoBarras (String), vencimento (String); implementa Pagavel com regras específicas para emissão/compensação.

- Classe CartaoCredito: atributos numeroCartao (String), nomeTitular (String), limiteDisponivel (double); implementa Pagavel verificando o limite antes de autorizar pagamento.
- Classe TransferenciaBancaria: atributos bancoOrigem (String), bancoDestino (String), contaOrigem (String), contaDestino (String); implementa Pagavel com validações de conta.

O que deverá ser construído na classe principal:

a) Criar um Boleto com codigoBarras = "1234567890" e vencimento = "2025-05-10".
b) Criar um CartaoCredito com numeroCartao = "1111 2222 3333 4444", nomeTitular = "João Silva", limiteDisponivel = 500000.0.
c) Criar uma TransferenciaBancaria com bancoOrigem = "Banco A", bancoDestino = "Banco B", contaOrigem = "0001-01", contaDestino = "0002-02".
d) Reunir as três instâncias em uma colecção do tipo Pagavel e, para cada elemento, chamar processarPagamento() com um mesmo valor, por exemplo 10000.0.
e) Apresentar mensagens distintas para cada tipo de pagamento, evidenciando a implementação específica do método comum definido na interface.

Exercício 33 – Operação Bancária com Exceção

Descrição do Problema:

Modelar operações de depósito e saque em uma conta corrente, com lançamento e tratamento de exceções em situações inválidas.
Especificação das classes:

- Classe ContaCorrente: reutiliza atributos numero (String), titular (String), saldo (double), limite (double).
- Método depositar(double valor) que lança uma exceção apropriada se valor for menor ou igual a zero.

- Método sacar(double valor) que lança uma exceção personalizada (por exemplo SaldoInsuficienteException) se o valor solicitado exceder saldo + limite.

O que deverá ser construído na classe principal:

a) Criar uma conta com numero = "CC003", titular = "José Pedro" e limite compatível, por exemplo limite = 500.0.
b) Realizar um depósito válido, por exemplo 1000.0, verificando o aumento do saldo.
c) Tentar efetuar um saque com valor dentro do limite combinado de saldo e limite, por exemplo 1200.0, confirmando o sucesso da operação.
d) Tentar efetuar um saque com valor superior ao permitido, por exemplo 5000.0, sendo necessário tratar a exceção SaldoInsuficienteException através de um bloco de tratamento adequado.
e) Ao final, apresentar o saldo da conta após as operações bem-sucedidas e exibir mensagens descritivas nos casos em que exceções forem lançadas, reforçando o controlo de erro na lógica de negócio.

Exercício 34 – Leitura de Arquivo de Alunos com Exceção de Formato

Descrição do Problema:

Modelar a leitura de um ficheiro de texto contendo registos de alunos, tratando tanto erros de acesso ao ficheiro como problemas de formato de dados.

Especificação das classes:

- Classe Aluno: atributos nome (String), numero (String), curso (String).
- Classe LeitorArquivoAlunos: método carregarDeArquivo(String caminho) que devolve uma lista de Aluno.
- Formato esperado de cada linha do ficheiro: "nome;numero;curso".

- Tratamento de exceções para problemas de I/O (ficheiro inexistente ou inacessível) e para linhas mal formadas (por exemplo, com menos campos), podendo usar uma exceção específica de formato.

O que deverá ser construído na classe principal:

a) Definir um caminho de ficheiro de teste, por exemplo "alunos.txt", contendo algumas linhas com registos válidos e eventualmente alguma linha com formatação incorreta.
b) Criar uma instância da classe responsável pela leitura e chamar o método carregarDeArquivo, passando o caminho definido, recorrendo a um bloco de tratamento de exceções para capturar problemas de I/O.
c) Introduzir tratamento específico para exceções de formato de linha, permitindo identificar e sinalizar linhas que não obedecem ao padrão "nome;numero;curso".
d) Caso a leitura seja bem-sucedida, percorrer a lista de alunos devolvida e apresentar nome, numero e curso de cada um, confirmando a correcta conversão do texto em objetos Aluno.
e) Exibir mensagens claras para os casos em que o ficheiro não seja encontrado ou contenha linhas mal formadas, reforçando o uso de exceções para lidar com situações anómalas.