

Universidade de Fortaleza

Programação Orientada a Objetos

Prof. Rommel Dias

AV1

Aula: Transição para o Java (parte I)

Explicação. Apresentar tipos de variáveis, condicionais e repetição.

Aula: Transição para o Java (parte II)

Explicação. Vetores e algoritmos fundamentais.

Aula: Transição para o Java (parte III)

Exercício. Atividade com linguagem Java.

Aula: Classes - atributos e métodos (parte I)

Explicação. Objetos, classes, atributos e métodos.

Exercício. Criar a classe Pessoa que tem como atributos: nome, ano de nascimento, peso, altura, dentre outros. Implementar métodos para calcular o IMC, idade, quantidade de água que a pessoa necessita beber por dia. Implementar método “mostra” (void).

Explicação. Comparação de objetos.

Aula: Classes - atributos e métodos (parte II)

Exercício. Criar a classe Conta que possui o CPF do titular da conta, o número representativo do banco e o saldo. Implementar método que retorna a bonificação do correntista. A bonificação é 10% do valor do saldo. Implementar métodos que realizam o saque de um valor e o depósito de um valor. Implementar método para mostrar os atributos.

Exercício. Modificar a classe e incluir um atributo que indica se o correntista é um cliente especial ou não. Refazer o método que retorna a bonificação do correntista. A bonificação é 10% do valor do saldo se o cliente for especial. Caso não seja, é de 5% do valor do saldo.

Exercício. Implementar método que realiza a transferência de valor de uma conta para outra.

Aula: Classes - vetores de objetos

Introduzir vetores de objetos.

Exercício. Criar a classe Produto, que possui um código (String), um peso e um valor (em reais). Implementar um método para mostrar as informações de um produto. Criar um vetor que armazena n produtos. Implementar algoritmos para:

- contar quantos produtos tem peso maior que 10 e valor menor que R\$ 50,00;
- calcular a média dos pesos;
- capturar o produto mais leve e mostrar as informações dele chamando o método implementado;
- capturar o produto de maior valor e mostrar as informações dele chamando o método implementado;;
- calcular a média dos valores cujo peso do produto é maior que 10.

Aula: Classes - construtores e encapsulamento (parte I)

Introduzir o conceito de encapsulamento e construtor através da classe Conta.

Exercício. Criar uma classe para representar uma pessoa, com os atributos privados de nome, CPF, ano de nascimento e altura. Crie os métodos públicos necessários para gets e sets e também um método para mostrar todos dados de uma pessoa. Crie um método para calcular a idade da pessoa.

Exercício. A fim de representar empregados em uma firma, crie uma classe chamada Empregado que inclui as três informações a seguir como atributos: nome, sobrenome e salário mensal. Sua classe deve ter um construtor que inicializa os três atributos. Forneça um método set e get para cada atributo. Se o salário mensal não for positivo, configure-o como 0. Na classe principal, crie dois objetos do tipo Empregado e exiba o salário anual de cada um. Na classe Empregado, implemente o método forneceAumento, que aumenta em 10% o salário de um empregado. Faça testes na classe principal.

Exercício. Criar uma classe chamada Invoice que possa ser utilizada por uma loja de informática para representar uma fatura de um item vendido. Uma fatura deve incluir as seguintes informações: o identificador (String) do item faturado, a quantidade comprada do item e o preço unitário do item. Ao utilizar um construtor, inicialize um objeto do tipo Invoice. Se a quantidade não for positiva, ela deve ser configurada como 0. Se o preço do item não for positivo ele deve ser configurado como 0.0. Além disso, forneça um método que calcula o valor da fatura, além de gets e sets.

Aula: Classes - construtores e encapsulamento (parte II)

Uso do this. Aula de exercícios.

Exercício. Escrever a classe Ponto2D que representa um ponto no plano cartesiano. Além dos atributos por você identificados, a classe deve oferecer as seguintes características: (i) atributos privados do ponto; (ii) métodos gets e sets; (iii) método que compara se um ponto é igual a outro; (iv) método que calcula a distância euclidiana de um ponto até outro.

Aula: Classes - construtores e encapsulamento (parte III)

Exercício. Criar uma classe denominada Elevador para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual (térreo = 0), total de andares no prédio, capacidade do elevador (número máximo de pessoas) e quantas pessoas estão presentes nele. Além de métodos gets e sets, a classe deve disponibilizar os seguintes componentes:

- construtor: que deve receber como parâmetros a capacidade do elevador e o total de andares no prédio (os elevadores sempre começam no térreo e vazio);
- entra: para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço);
- sai: para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);
- sobe: para subir um andar (não deve subir se já estiver no último andar);
- desce: para descer um andar (não deve descer se já estiver no térreo).

Exercício. Crie a classe Data para representar datas observando os pontos a seguir.

- represente uma data usando três atributos: o dia, o mês e o ano;
- implemente métodos gets e sets;
- implemente um método que retorna a representação da data como String. Considere que a data deve ser formatada mostrando o dia, o mês e o ano separados por barra (/). Pesquise na internet, caso necessário, uma forma de transformar inteiros em Strings;
- implemente um método para checar se uma data é igual à outra;

- uma das operações que podemos efetuar com datas é a comparação para verificar se uma data ocorre antes de outra. Escreva o método `vemAntes` que receba como argumento outra instância da classe `Data`, retornando `true` se a data da classe vier antes da passada como argumento e `false` caso contrário. Se as datas forem exatamente iguais, o método deve retornar `true`;
- na classe principal, inicialize um objeto `Data` e faça uma chamada aos métodos implementados para verificar o correto funcionamento.

Exercício. Uma turma é constituída por de alunos. Implemente a classe `Turma`, que tem como atributos: um vetor que possui o nomes dos alunos; um vetor que possui a idade dos alunos; e um vetor que possui a média dos alunos na disciplina. A classe deve possuir construtor e métodos `get/set` associados aos atributos. Implemente também os seguintes métodos:

- `idadeMaisVelho`, que retorna a idade do(a) aluno(a) mais velho(a);
- `nomeMaisNovo`, que retorna o nome do(a) aluno(a) mais novo(a).
- `numeroAprovados`, que retorna o número de alunos aprovados. Um aluno é aprovado se possui média igual ou superior a 7,0.

Na função principal, inicialize um objeto do tipo `turma` e faça uma chamada a cada um dos métodos.

Exercício. Uma imobiliária trabalha com dois tipos de imóveis (casa e apartamento). Todo e qualquer imóvel possui um tipo (1 para casa e 2 para apartamento), um proprietário (nome), um tamanho (em metros quadrados), um identificador se está em área nobre ou não, o ano em que o imóvel foi construído, e um valor de compra.

O cálculo do IPTU de um imóvel pode ser feito por meio das seguintes regras:

Tipo	Tamanho	IPTU (% sobre valor de compra)
Apartamento	> 100	0,05
Apartamento	≤ 100	0,02
Casa	> 100	0,03
Casa	≤ 100	0,01

Também manipulando os dados fornecidos, pode-se calcular um possível preço de venda usando as seguintes condições:

Área nobre	Tamanho	Valor de venda
Sim	> 100	180% o valor da compra
Sim	≤ 100	110% o valor da compra
Não	> 100	120% o valor da compra
Não	≤ 100	105% o valor da compra

A partir disso:

- implemente a classe `Imovel` com os devidos atributos, construtor, métodos `gets` e `sets`;
- implemente um método para calcular e retornar o valor do IPTU do imóvel;
- implemente um método para calcular e retornar o valor de venda do imóvel;
- implemente um método para calcular a idade de um imóvel;
- implemente um método para mostrar as informações (atributos) do imóvel;

Na classe principal:

- declare e inicialize um vetor com informações para n imóveis, onde n é um dado de entrada (utilize o Scanner para receber todos os dados);
- mostre os IPTUs e os valores de venda dos n imóveis;
- mostre os nomes de todos os proprietários dos imóveis com idade superior a 10 anos;
- mostre a quantidade de apartamentos e de casas cadastradas;
- mostre a média do tamanho dos apartamentos localizados em área nobre;
- mostre o menor IPTU de uma casa cadastrada em uma área que não seja nobre;
- mostre o tamanho do imóvel com menor valor de venda.

Aula: Classes - relacionamento entre classes (parte I)

Exercício. Todo cliente, por padrão, tem um CPF que o identifica e um endereço. Implemente a classe Cliente com atributos, regras de encapsulamento e o método “imprimir”, que exibe as informações de um cliente.

Toda conta bancária, por sua vez, é associada a um cliente. Além disso, toda conta tem um saldo que não pode ser negativo. Implemente a classe Conta com os devidos atributos, regras de encapsulamento e método “imprimir”.

Instancie na classe principal um objeto do tipo Conta.

Exercício. Com relação ao exercício anterior, implemente a classe Operacao. Toda operação tem um tipo (D – depósito ou R – retirada) e um valor associado à operação. Siga regras de encapsulamento.

Na classe Conta, implemente o método fazOperacao, que realiza uma operação e atualiza o saldo da conta.

Realize n operações sobre uma conta na classe principal.

Aula: Classes - relacionamento entre classes (parte II)

Exercício. Implemente a classe Aluno. Cada aluno tem nome, ano de nascimento e média global. Implemente também o método “imprimir” para exibir o valor dos atributos e siga regras de encapsulamento.

Implemente a classe Turma. Uma turma tem um conjunto de alunos e um curso. Siga regras de encapsulamento. Na classe Turma, implemente: um método que retorna o aluno que possui a maior média; um método que retorna o aluno mais novo.

Instancie um objeto Turma na classe principal e faça uma chama aos métodos.

Exercício. Faça um programa de agenda telefônica com as classes Agenda e Contato. Cada contato tem nome, telefone e um indicador se é um contato favorito ou não. Cada agenda tem um conjunto de contatos.

Na classe Agenda, implemente um método para retornar a quantidade de contatos favoritos. Implemente também um método que retorna quantos contatos que começam com a letra “R” (ou “r”) a Agenda possui.

Na classe principal, inicialize um objeto do tipo Agenda e faça uma chamada ao método implementados na classe.

Exercício. Escreva a classe Voo, que armazena e realiza operações de um voo. Cada voo tem uma data de operação (implementar classe Data da Aula 07), uma quantidade de assentos e um vetor que representa os assentos do avião. O vetor indica se um assento está reservado ou não.

A classe deve possuir:

- construtor: tem como parâmetro uma data e o número de assentos. A inicialização de cada assento deve ser feita como desocupado;
- método “verifica”: retorna um indicativo se um assento está ocupado ou não;
- método “ocupa”: reserva determinado assento do voo;
- método “vagas”: retorna o número de cadeiras vagas disponíveis no voo.
- método “imprimir”: mostra a data do voo e as cadeiras ainda disponíveis.

Faça chamadas aos métodos na função principal para verificar o funcionamento das operações.

Exercício. Uma loja de departamento realiza a venda de produtos diversos. Cada produto tem um identificador (código com letras e números), um nome, um valor e uma quantidade disponível no estoque da loja. Implemente a classe Produto com essas informações, além de construtor, gets e sets, e método “imprimir”.

Implemente também a classe Loja, que possui um conjunto de produtos. Na classe, implemente os métodos:

- “vender”: que realiza a venda de uma unidade de um produto;
- “abastecer”: que realiza o abastecimento de uma unidade de um produto;
- “valorEstoque”: que computa o valor total do estoque da loja.

Instancie um objeto do tipo Loja na classe principal e faça uma chamada aos métodos implementados.

Aula: Herança simples

Conceito de superclasse e subclasse (hierarquia); as palavras-chaves extends, super e protected; a anotação @Override.

Exercício. Implemente a classe Funcionario, que possui como atributos o nome, o CPF e o salário do funcionário de uma empresa. Siga regras de encapsulamento. Implemente também um método que calcula e retorna o bônus de um funcionário. O bônus é de 5% do salário.

Implemente a classe Presidente. A classe deve possuir os mesmos atributos e métodos da classe Funcionario, mas um dado a mais: a quantidade de ações que ele tem da empresa.

Instancie objetos do tipo Funcionario e Presidente. Para cada objeto, faça uma chamada ao método que calcula o bônus e mostre em tela.

Sabe-se agora que o bônus de um presidente é de 10% do salário. Reimplemente o método na classe Presidente e refaça a chamada na classe principal.

Exercício. Uma universidade está fazendo o cadastro de estudantes para armazenamento no sistema.

Implemente a classe Estudante. Todo(a) estudante possui: a matrícula; o ano de ingresso na universidade; e o curso que faz. Siga regras de encapsulamento. Na instituição em que os(as) estudantes estão vinculados(as), há a

opção de tirar cópias em determinados locais do campus. O preço da cópia de uma folha é de R\$0,10. Implemente um método que retorna o valor gasto com a cópia de um material.

Sabe-se que estudantes presentes na universidade podem adquirir bolsas vinculadas a projetos de pesquisa. Implemente a classe Bolsista. Dentre os atributos, um(a) bolsista possui o valor (em R\$) que recebe com a bolsa. Siga regras de encapsulamento.

Como bolsistas são vinculado(as) a projetos de pesquisa, há um desconto que a universidade fornece com cópia de materiais, sendo o valor da folha reduzido para R\$0,07. Implemente um método que retorna o valor gasto com a cópia de um material.

Na classe Bolsista, implemente um método que retorna quantas cópias ele pode tirar com o valor da bolsa que recebe.

Instancie um objeto do tipo Estudante e outro do tipo Bolsista. Faça uma chamada aos métodos implementados.

Aula: Herança múltipla

Conceito de interface.

Exercício. Implemente a interface Seguranca. Ela deve possuir assinatura para o método booleano verificaSenha(senha), além do método mensagemSucesso() e mensagemFracasso() que mostra uma mensagem de sucesso/falha após o usuário digitar a senha.

Implemente a classe Conta. A classe deve possuir como atributos: CPF, senha e saldo do(a) usuário(a). Como métodos, deve possuir “saca(valor, senha)” e “deposita(valor, senha)”. O ato de sacar ou depositar o valor só é realizado se a senha estiver correta. Portanto, a classe deve possuir o método verificaSenha(senha) para tal tarefa e mensagemSucesso() e mensagemFracasso() para informar do sucesso/fracasso da operação.

Implemente a classe Email. A classe deve possuir como atributos: nome do(a) usuário(a), e-mail e senha. Como métodos, deve possuir o método verificaSenha(senha), mensagemSucesso() e mensagemFracasso() para infor-

mar do sucesso/fracasso do acesso.

Exercício. 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 retângulo 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.

Aula: Polimorfismo (parte I)

Introduzir o conceito de polimorfismo (várias formas); classes abstratas e métodos abstratos; instanceof.

Exercício. Criar a classe Animal. Todo animal tem um nome e uma idade. Siga regras de encapsulamento.

Um cachorro é um animal. Implemente a classe Cachorro que, além dos atributos relacionados a um animal, possui um indicador se ele corre ou não.

O bicho preguiça também é um animal. Implemente a classe Preguica que, além dos atributos relacionados a um animal, possui um indicador se ele escala ou não.

Em cada classe, implemente um método que mostra o som emitido por cada um dos animais.

Na classe principal, instancie um objeto do tipo Cachorro e outro do tipo Preguica. Chame o método que mostra o som emitido pelos animais.

Declare um objeto do tipo animal e instancie de forma polimórfica. Faça uma chamada ao método para emissão de som.

Exercício. Ainda de acordo com o exercício anterior, use uma estrutura de dados (vetor) para armazenar, de forma conjunta, objetos do tipo Cachorro e Preguica. Calcule a média da idade dos animais cadastrados.

Aula: Polimorfismo (parte II)

Exercício. Um imóvel é uma propriedade e pode fazer menção a um bem imobiliário, sendo uma alternativa para investidores que desejam adquiri-lo e vendê-lo após alguns meses ou anos.

Você é o(a) responsável por administrar um sistema de gerenciamento de imóveis. Seguindo regras de encapsulamento, implemente a classe Imovel sabendo que qualquer imóvel tem uma área (em metros quadrados), uma quantidade de quartos e um preço de compra.

Casas e apartamentos são exemplos de imóveis. Implemente as classes Casa e Apartamento. A classe Casa, além dos atributos relacionados a um imóvel, tem um indicador se há quintal ou não, enquanto a classe Apartamento tem um indicador se há piscina na área externa ou não. Siga regras de encapsulamento.

Casas que possuem quantidade de quartos superior a 4 ou que têm quintal são vendidas com 30% a mais sobre o valor de compra; do contrário, são vendidas com 15% a mais. Na classe Casa, faça um método que calcule e retorne o preço de venda de uma casa.

Apartamentos que possuem área maior que 300 metros quadrados e que têm piscina são vendidos por 50% a mais sobre o valor de compra; do contrário, são vendidos com 10% a mais. Na classe Apartamento, faça um método que calcule e retorne o preço de venda de um apartamento.

Seu sistema de gerenciamento de imóveis deve permitir expansão para futuras classes (de imóveis) que venham a ser implementadas, como casas de praia, casas de campo, escritórios, dentre outras. Todas elas terão um método que calcula e retorna o preço de venda de um imóvel. Garanta que seu programa em Java assegure o programador da implementação de tal método.

Na classe principal, instancie aleatoriamente um vetor com n imóveis, sejam eles do tipo Casa ou Apartamento. Faça esse vetor ser um atributo da classe Imobiliaria, relacionada a uma imobiliária que comercializa esse conjunto de imóveis.

Na classe Imobiliaria, implemente um método que:

- calcula e retorna a média do preço de compra dos imóveis;
- calcula e retorna a quantidade de casas;
- calcula e retorna a quantidade de apartamentos;
- calcula e retorna o imóvel de maior preço de venda.

Exercício. Ingressos são comercializados a todo momento em Fortaleza, principalmente em bilheterias virtuais. Seja para cinema, show, micareta ou peças de teatro, é possível adquirir um ingresso online.

São comercializados diferentes tipos de ingressos. Eles podem ser VIP ou não. Podem ser meia entrada ou não. Podem ser também de lotes distintos (1, 2, 3, ...).

Implemente uma classe chamada Ingresso. A classe deve possuir os seguintes atributos: nome (nome do evento); ehMeia (indica se é meia entrada ou não); preço (valor do ingresso); e lote (número do lote).

Só há dois tipos de ingressos comercializados: VIP e comum. Implemente as classes IngressoVip e IngressoComum. Ambas as classes devem possuir o método obrigatório de todo ingresso: calculaReembolso, que retorna um reembolso.

- Para ingressos comuns, o reembolso é de 5% do valor se a entrada é inteira ou se o lote é o primeiro. Do contrário, o reembolso é de 3% do preço;

- Para ingressos VIPs, o reembolso é de 10% do valor se a entrada é inteira e lote é o primeiro. Do contrário, o reembolso é de 6% do preço.

Implemente o método `calculaReembolso` em cada uma das subclasses. Utilize ele como cabeçalho na classe abstrata `Ingresso`.

Na classe principal, declare e receba n ingressos e armazene eles em um único vetor. Mostre em tela os reembolsos associados a todos eles.

Exercício. Uma concessionária vende veículos de transporte, e o administrador da concessionária deseja que os dados dos veículos sejam cadastrados no sistema para um bom planejamento das informações e do estoque. Você foi o(a) encarregado(a) dessa tarefa.

Crie a classe `Veiculo`. Sabe-se que todo veículo possui um indicador se a cor é metálica, o valor de fabricação (isto é, o quanto ele custou para ser fabricado) e a quantidade disponível em estoque. Além disso, todo veículo tem um método que calcula e retorna o valor de venda.

Dentre os veículos comercializados pela concessionária, há carros, motos e vans. Seguindo regras de encapsulamento, crie as classes `Carro`, `Moto` e `Van`.

Todo carro possui um número de portas, além de um indicador se possui câmbio automático. O valor de venda segue as regras abaixo:

Portas	Automático	% sobre fabricação
2	não	17%
2	sim	35%
4	não	31%
4	sim	55%

Toda moto possui uma quantidade de cilindradas, e o preço de venda de uma moto segue as regras abaixo:

Toda van possui uma quantidade de assentos disponíveis e uma capacidade máxima de peso (kg) que ela suporta. O preço de venda de uma van segue as regras abaixo:

Cilindradas	Cor metálica	% sobre fabricação
< 500	não	5%
< 500	sim	11%
≥ 500	não	25%
≥ 500	sim	27%

Assentos	Peso suportado	% sobre fabricação
< 7	< 1000	12%
< 7	≥ 1000	19%
≥ 7	< 1000	35%
≥ 7	≥ 1000	47%

Todos os veículos possuem um valor de seguro que equivale a 5% o valor da venda do veículo. Identifique a classe ideal para criar um método que calcula e retorna o valor do seguro de um veículo.

Instancie aleatoriamente um vetor com carros, motos e vans. Calcule o maior valor de fabricação associado a um veículo.

Exercício. Uma loja de departamento vende produtos eletrônicos. O diretor da loja deseja que os dados dos produtos sejam cadastrados no sistema para um bom planejamento das informações e do estoque. Você foi o(a) encarregado(a) dessa tarefa.

Cada produto possui um nome associado ao modelo, um preço e um tempo de garantia (em meses). Crie a classe Produto seguindo regras encapsulamento.

A loja trabalha majoritariamente com dois tipos de produtos: computadores e celulares. Crie as classes Computador e Celular sabendo que um computador tem um indicador se vem impressora ou não, enquanto um celular tem um indicador se vem carregador ou não.

Cada produto possui um método que calcula e retorna o valor de um desconto no ato da compra a partir de uma forma de pagamento (1 ou 2). Para computadores:

Para celulares:

Crie a classe Loja. A classe possui como atributo um conjunto de produtos. Na classe, crie os 2 métodos abaixo:

- `qntCelCarregador`, que retorna a quantidade de celulares que tem car-

Impressora	Pagamento	% desconto sobre preço original
sim	1	10%
sim	2	7%
não	1	5%
não	2	não tem

Carregador	Pagamento	% desconto sobre preço original
sim	1	12%
sim	2	10%
não	1	7%
não	2	não tem

regador;

- computadorBarato, que retorna o computador de menor preço.

Instancie um objeto do tipo Loja e faça uma chamada aos métodos implementados na classe.

Aula: Estruturas avançadas

Introdução da estrutura ArrayList; métodos associados à estrutura ArrayList.

Exercício. Crie um ArrayList de números inteiros. Efetue operações de adição, remoção, busca, modificação e outros métodos.

Exercício. Crie um objeto chamado Cliente com os atributos: id (inteiro que determina o identificador de uma matrícula), nome, ano de nascimento, telefone.

Suponha, agora, que uma academia trabalha com um conjunto de clientes. Crie uma classe chamada Academia, que tem como atributos uma lista de clientes e o nome da academia. A classe Academia, além de gets e sets, deve ter métodos para:

- cadastrar um novo cliente na academia;
- remover um cliente da academia a partir de um id (cliente resolveu não se matricular novamente);
- retornar o número de clientes cadastrados na academia;
- retornar os clientes que nasceram após o ano 2000;
- remover clientes que nasceram antes de 1990;
- outros métodos.

Instancie um objeto do tipo Academia. Toda academia começa com nenhum cliente. Cadastre clientes na academia a partir do método de cadastro. Remova alguns a partir do método remoção. Mostre o número de clientes até então matriculados.

Exercício. Represente um veículo inteligente. Todo veículo inteligente tem a velocidade atual (em quilômetros por hora) em um momento qualquer, o nível atual de combustível (em litros) no tanque em um momento qualquer, e o nível máximo de combustível (em litros) no tanque. Todo veículo começa parado e com o tanque cheio (atenção: o construtor dessa classe só

tem uma entrada). Nessa classe, implemente um método que realiza o ponto de partida do veículo a 10 km/h e um método que realiza a parada do veículo.

Um dos tipos de veículos inteligentes que você é responsável é o Aerospace. Alguns modelos desse tipo de veículo têm bagageiro e outros não. Outro tipo de veículo pelo qual você é responsável é o Motorpower, que pode ser SUV ou não. Represente-os de forma adequada.

Como os veículos inteligentes possuem autocontrole para aceleração/frenagem, eles aceleram/freiam sem a necessidade que uma pessoa acione o acelerador/freio, modificando então a velocidade do veículo.

- Para o Aerospace, se o veículo for compacto (isto é, não tiver bagageiro), a velocidade atual for menor que 30 km/h e ele se encontrar em uma área urbana, há um aumento de 50% sobre a velocidade atual; do contrário, há uma redução de 5% sobre a velocidade atual;
- Para o Motorpower, se o veículo não for SUV, ou velocidade atual for menor que 25 km/h ou ele não se encontrar em uma área urbana, há um aumento de 100% sobre a velocidade atual; do contrário, o veículo permanece com a mesma velocidade.

Represente essa dinâmica com o método “controlarVelocidade” na(s) classe(s) adequada(s).

Os veículos inteligentes também possuem autorreconhecimento sobre o nível de combustível, indicando se há a necessidade para o abastecimento. No caso do Aerospace, se o nível de combustível for menor que 20 litros, o tanque é completado por inteiro. No caso do Motorpower, se o nível de combustível for menor que 10 litros, preenche-se totalmente o tanque também. Represente essa dinâmica com o método “abastecer” na(s) classe(s) adequada(s).

Crie uma classe Administracao que realiza o monitoramento de veículos inteligentes e tem como atributo uma lista de veículos (seja Aerospace, seja Motorpower) inicialmente vazia. Implemente métodos para:

- adicionar um novo veículo inteligente na loja;
- remover um veículo dado como entrada (use o equals);
- retornar veículos que tem velocidade atual maior que um valor;
- remover veículos parados.

Aula: Exceções

Erros:

- tempo de compilação (ex: caracteres inválidos);
- tempo de execução (ex: indexação de arrays e objetos null).

Exceção: erro que ocorre em **tempo de execução**.

Tratamento de exceção: permite tratar erros de tempo de execução sem que o programa seja interrompido. Mais ainda, permite você **estruturar** a identificação de erros.

Palavras-chave: try e catch.

Obs.: assim como os erros, as exceções podem ser de vários tipos: ArithmeticException, ArrayIndexOutOfBoundsException, etc.

Exemplos de exceções com try/catch: momento Eclipse.

Obs.: pode-se associar mais de uma instrução catch a uma instrução try. Cada catch deve capturar um tipo de exceção diferente.

Exemplos de exceções com try/catches: “tente fazer algo. Caso não dê certo, vá imediatamente para catch” (momento Eclipse).

Obs.: e quando não se sabe o tipo de exceção?

Palavra-chave: finally (o que vem em finally é sempre executado, independente do fluxo do programa chegar de try ou catch).

Aula: Arquivos

Leitura e escrita de arquivos.

Escrita de dados:

Uma eleição possui 3 candidatos (1, 2 e 3). Vários eleitores foram às urnas para votar em um deles.

Implemente a classe Eleitor. A classe tem como atributos o CPF do eleitor, o ano de nascimento e o candidato em quem ele(a) votou. Implemente construtor, gets, sets e o método toString().

Crie uma classe principal chamada Escrita. Declare uma lista de eleitores, inicialmente vazia. Receba um conjunto **indeterminado de eleitores**. O recebimento de dados é interrompido quando o voto do eleitor for diferente de 1, 2 e 3.

Salve os dados dos eleitores no arquivo “eleicao.txt” por meio de chamadas ao método toString().

Leitura de dados:

Faça o processo inverso. Crie uma classe principal chamada Leitura. Declare uma lista de eleitores, inicialmente vazia. Receba um conjunto **indeterminado de eleitores** por meio de uma leitura ao arquivo “eleicao.txt”.

Mostre em tela quantos eleitores foram recebidos como entrada.

DESAFIOS

Desafios

Desafio. Escreva um programa completo para jogar o jogo da velha. Para tanto crie uma classe `JogoDaVelha`, que possui como atributo um array bi-dimensional 3×3 e um resultado final. Detalhes de implementação:

- o construtor da classe deve inicializar o array completamente vazio;
- implemente um método para exibir o array;
- implemente o método `realizarJogada`, que recebe como entrada o identificador de um jogador (1 ou 2) e a posição na qual ele escolheu jogar;

Realize jogadas na classe principal.

Desafio. Utilizando conceitos de orientação a objeto (livre), implemente o jogo batalha naval.

Entrada de dados para cada jogador:

- um tabuleiro (matriz) de tamanho 10×10 ;
- um conjunto de navios de dimensões:
 - 1×2 (submarino): 4 navios;
 - 1×3 (contratorpedeiros): 3 navios;
 - 1×4 (navios tanque): 2 navios;
 - 1×5 (porta-aviões): 1 navio.
- os navios devem ser dispostos, de forma aleatória e sem sobreposição, no tabuleiro;
- considere que os navios possam ser rotacionados (2×1 , 3×1 , 4×1 e 5×1).

Dinâmica do jogo:

- dois jogadores jogam o batalha naval;
- em cada jogada, um jogador escolhe uma posição (linha e coluna) para atacar o tabuleiro do oponente;

- posições repetidas não devem ser consideradas;
- o jogo deve ser executado de forma aleatória.

Arquivos:

- em arquivos, chamados “ataques1.txt” e “ataques2.txt”, escreva as posições de ataque de cada jogador (linhas e colunas).
- em um arquivo, chamado “saida.txt”, escreva qual jogador venceu o jogo e o número de ataques que cada um fez.

Desafio. O problema de carregamento de contêiner tem como entrada um único contêiner, com comprimento C , largura L e altura A , todas em metros.

O problema também tem como entrada n tipos de caixas. Cada tipo de caixa tem um tipo t , um comprimento c , uma largura l , uma altura a e uma quantidade q de caixas disponíveis. As dimensões podem ser rotacionadas ou não.

Você deverá criar uma classe chamada Objeto3D para representar objetos tridimensionais. A classe Container e a classe Caixa também deverão ser criadas, herdando certos atributos de Objeto3D. Em cada subclasse, se necessário, adicione atributos e métodos específicos.

O objetivo do problema empacotar as caixas dentro do contêiner de tal forma que o aproveitamento do espaço interno seja o maior possível, isto é, o volume das caixas empacotadas dividido pelo volume do contêiner deve ser o mais próximo de 1 possível.

Os dados de entrada do problema estarão em um arquivo .txt com o seguinte padrão:

```
587 233 220
3
1 108 1 76 0 30 1 40
2 110 0 43 1 25 1 33
3 92 1 81 1 55 1 39
```

A primeira linha especifica as dimensões do contêiner, isto é, $C = 587$, $L = 233$ e $A = 220$. A segunda linha traz a quantidade de tipos de caixas $n = 3$. As demais 3 linhas determinam os dados de cada tipo de caixa.

Como referência, considere a linha do primeiro tipo de caixa, isto é:

1 108 1 76 0 30 1 40

Nela, o número 1 indica o tipo da caixa, cujo comprimento $c = 108$, largura $l = 76$, altura $a = 30$ e quantidade $q = 40$.

Após cada dimensão, há um indicador binário. O número 1 significa que o valor que o antecede pode ser colocado na altura da caixa. Por exemplo: após o número 108, temos o indicador 1. Portanto, a caixa pode ser rotacionada de tal forma que 108 seja a altura dela. Portanto, a cada do tipo 1 pode ter os padrões:

$$c = 76, l = 30, a = 108$$

$$c = 30, l = 76, a = 108$$

Observe que o número 76 tem o indicador 0 à direita. Portanto, 76 não pode ser considerado na posição vertical. No entanto, o valor 30 tem o indicador 1 à direita. Dessa forma, tem-se os seguintes padrões:

$$c = 108, l = 76, a = 30$$

$$c = 76, l = 30, a = 30$$

Portanto, a caixa do tipo 1 pode assumir 4 formas para o empacotamento dentro do contêiner:

$$c = 76, l = 30, a = 108$$

$$c = 30, l = 76, a = 108$$

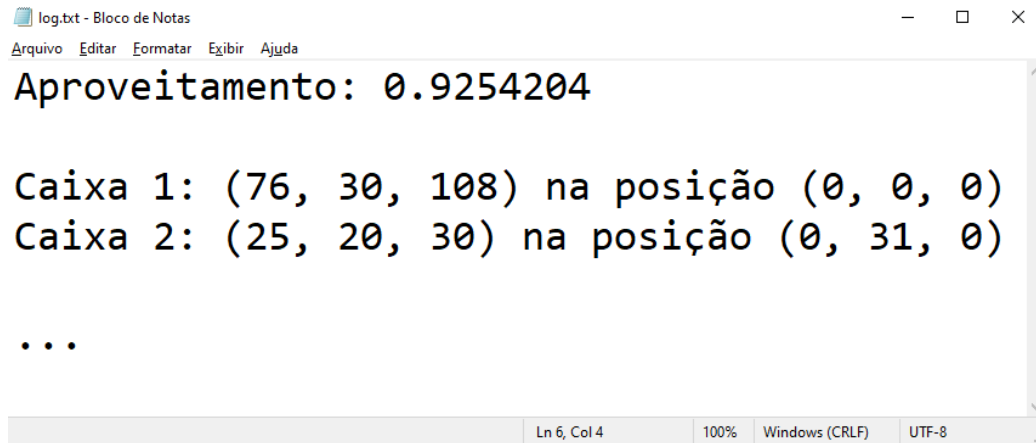
$$c = 108, l = 76, a = 30$$

$$c = 76, l = 108, a = 30$$

Cada equipe estará livre para escolher uma estratégia de empacotamento de caixas. No entanto, duas restrições devem ser seguidas no ato de empacotamento:

- obviamente, as caixas deverão estar contidas dentro do contêiner, isto é, uma caixa de comprimento $c = 200$ jamais poderia se encontrar dentro de um contêiner de comprimento $C = 150$;
- uma caixa posicionada acima de outra caixa deve ter a base completamente suportada pela caixa de baixo.

A saída do programa deverá estar em um arquivo chamado *log.txt*. Esse arquivo deverá possuir o aproveitamento do espaço do contêiner (número entre 0 e 1), os dados de cada caixa empacotada e a posição do contêiner onde ela foi instalada. Por exemplo:



```
log.txt - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
Aproveitamento: 0.9254204
Caixa 1: (76, 30, 108) na posição (0, 0, 0)
Caixa 2: (25, 20, 30) na posição (0, 31, 0)
...
Ln 6, Col 4  100%  Windows (CRLF)  UTF-8
```