Exercícios propostos de fixação

Exercícios sobre classes e objetos em Java

Exercício 1
Desenvolver uma classe Java chamada **Apolice** com os seguintes atributos: *nomeSegurado*, *idade* e *valorPremio*. A classe **Apolice** deverá conter os seguintes métodos:

Método	Descrição
imprimir()	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe **Apolice**.
calcularPremioApolice()	Este método não retorna valor e deverá calcular o valor do prêmio seguindo as seguintes regras: caso a idade seja maior ou igual a 18 e menor ou igual a 25 anos, use a fórmula: valorPremio += (valorPremio*20)/100. Quando a idade for superior a 25 e menor ou igual a 36 anos, use a fórmula: valorPremio += (valorPremio*15)/100. Quando a idade for superior a 36, use a fórmula: valorPremio += (valorPremio*10)/100.
oferecerDesconto()	Este método não retorna valor, mas recebe o parâmetro cidade, que irá conter o nome da cidade para o cálculo do desconto. Caso a cidade seja Curitiba, dê um desconto no valor do prêmio de 20%. Caso seja Rio de Janeiro, dê um desconto no valor do prêmio de 15%. Caso seja São Paulo, dê um desconto no valor do prêmio de 10%. Caso seja Belo Horizonte dê um desconto no valor do prêmio de 5%. Para realizar a comparação de strings neste exercício utilizar o método **equals()**.

Exercício 2
Desenvolver uma segunda classe Java chamada **PrincipalApolice** com a seguinte estrutura:

- Criar o método **main**() conforme padrão da linguagem Java. Nesse método, criar um objeto da classe **Apolice**. Para cada atributo da classe atribuir um valor coerente.
- Executar o método **imprimir**() e analisar o que será impresso na tela.
- Em seguida, executar o método **calcularPremioApolice**().
- Executar o método **imprimir**() novamente e analisar o que será impresso na tela.
- Executar o método **oferecerDesconto**() passando como parâmetro a cidade de Curitiba.
- Executar o método **imprimir**() novamente e analisar o que será impresso na tela.

Exercício 3
Desenvolver uma classe chamada **Acampamento** com os seguintes atributos: *nome*, *equipe* e *idade*. Em seguida, implementar os seguintes métodos:

Método	Descrição
imprimir()	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe **Acampamento** na tela.
separarGrupo()	Este método não retorna valor e deverá verificar as seguintes condições: se a idade estiver entre 6 e 10 anos, atribuir A ao atributo **equipe**; se a idade estiver entre 11 e 20, atribuir B ao atributo **equipe**; se a idade for superior a 21 anos, atribuir C ao atributo **equipe**.

Exercício 4
Desenvolver uma segunda classe Java chamada **PrincipalAcampamento** com a seguinte estrutura:

- Criar o método **main**() conforme o padrão da linguagem Java.
- Criar um objeto da classe **Acampamento** e atribuir valores a seus atributos.
- Executar o método **imprimir**() e analisar o que será exibido na tela.

- Executar o método **separarGrupo**().
- Executar o método **imprimir**() novamente e analisar o que será exibido na tela.

Exercício 5
Desenvolver uma classe chamada **Computador** com os seguintes atributos: *marca*, *cor*, *modelo*, *numeroSerie* e *preco*. Implementar os seguintes métodos:

Método	**Descrição**
imprimir()	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe **Computador**.
calcularValor()	Este método não retorna valor e deverá verificar as seguintes condições: caso a marca seja HP, acrescentar 30% ao preço; caso seja IBM, acrescentar 50% ao preço; caso seja qualquer outra, manter o preço original.
alterarValor()	Este método recebe um valor como parâmetro. Atribuir este valor ao atributo **preço**, caso o valor do parâmetro recebido seja maior que 0. Caso seja maior que 0, o método **alterarValor**() deverá, além de atribuir o valor ao atributo **preço**, retornar 1. Caso contrário, não atribuir o valor ao atributo **preço** e retornar 0.

Exercício 6
Desenvolver uma segunda classe Java chamada **PrincipalComputador** com a seguinte estrutura:
- Criar o método **main**() conforme padrão da linguagem Java.
- Criar um objeto da classe **Computador** e atribuir valores a seus atributos. Atribuir HP ao atributo marca.
- Executar o método **imprimir**() e analisar o que será exibido.
- Executar o método **calcularValor**().
- Executar o método **imprimir**() e analisar o que será exibido.
- Criar um segundo objeto da classe **Computador** e atribuir valores a seus atributos. Atribuir IBM ao atributo **marca** do novo objeto.
- Executar o método **calcularValor**() do novo objeto.
- Executar o método **imprimir**() do novo objeto e analisar o que será exibido.
- Executar para o novo objeto o método **alterarValor**() com um valor positivo.
- Verificar no método **main**() o retorno do método **alterarValor**() e mostrar a mensagem de "valor alterado" caso este retorne 1, e "valor NÃO alterado" caso retorne 0.
- Executar para o novo objeto o método **alterarValor**() com um valor negativo.
- Verificar no método **main**() o retorno do método **alterarValor**() e mostrar a mensagem de "valor alterado" caso este retorne 1, e "valor NÃO alterado" caso retorne 0.
- Executar para o novo objeto o método **imprimir**() e analisar o que será exibido na tela.

Exercício 7
Desenvolver uma classe Java chamada **ContaCorrente** com a seguinte estrutura:

Atributos: *conta*, *agencia*, *saldo* e *nomeCliente*.
Métodos:

Método	**Descrição**
imprimir()	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe **ContaCorrente**.
sacar()	Retorna valor1 caso o saque seja realizado ou 0 se não houver saldo suficiente na conta. Deverá receber como parâmetro o valor a ser sacado.
depositar()	Realizar o depósito do valor recebido como parâmetro. Não deve retornar valor.

Exercício 8
Desenvolver uma segunda classe Java chamada **PrincipalContaCorrente** com a seguinte estrutura:

- Criar um atributo da classe ContaCorrente para ser usado pelos métodos da classe para realizar saques e depósitos.

Obs: atenção, pois ao executar o programa só poderemos fazer um saque se já tivermos realizado um depósito.

Métodos:

Método	**Descrição**
main()	Implementá-lo conforme padrão da linguagem Java. O método **main**() deverá criar um loop para o usuário escolher entre as opções cadastrar, sacar, depositar ou consultar. Se for selecionado depositar, executar o método **execDeposito**(). Se for selecionada a opção sacar, executar o método **execSaque**(). Para a opção consultar, executar o método **execConsulta**(). Para a opção cadastrar, executar o método **execCadastro**().
execSaque()	Solicitar ao usuário que digite um valor e executar o método **sacar**() da classe **ContaCorrente** usando o atributo criado. Testar o retorno do método **sacar**(). Se for retornado 1, exibir "Saque realizado", caso contrário, exibir "Saque NÃO realizado".
execDeposito()	Solicitar ao usuário que digite um valor e executar o método **depositar**() da classe **ContaCorrente** usando o objeto criado anteriormente.
execConsulta()	Apresentar os atributos na tela executando o método **imprimir**() da classe **ContaCorrente**.
execCadastrar()	Solicitar que o usuário realize a leitura dos dados via teclado e em seguida realize a atribuição dos valores lidos do teclado aos atributos do objeto da classe **ContaCorrente**, criado como atributo dessa classe.

Exercício 9
Desenvolver uma classe Java chamada **Eleitoral** com a seguinte estrutura:

Atributos: *nome* e *idade*.

Métodos:

Método	**Descrição**
imprimir()	Seguir a mesma especificação dos demais métodos. O método **imprimir**() deverá executar o método **verificar**() como último comando.
verificar()	O método **verificar**() não retorna valor e nem recebe parâmetro. Deve exibir na tela mensagem de acordo com as seguintes condições: caso a idade seja inferior a 16 anos, exibir na tela "Eleitor não pode votar";para idade superior ou igual a 16 anos e inferior a 65, exibir na tela "Eleitor deve votar". Para idade superior a 65 anos, exibir a tela "Voto facultativo".

Exercício 10
Desenvolver a classe PrincipalEleitoral com a seguinte estrutura:

Atributo: *valor*

Método:

Método	**Descrição**
main()	Implementá-lo conforme padrão da linguagem Java. Criar um objeto da classe Eleitoral e atribuir valores aos parâmetros. Executar o método

	imprimir() e analisar os valores exibidos na tela.

Exercício 11
Desenvolver uma classe Java chamada Estoque com a seguinte estrutura:

Atributos: *nomeProduto*, *valor* e *quantidade*.

Métodos:

Método	**Descrição**
imprimir()	Seguir a mesma especificação dos demais métodos.
verificarDisponibilidade()	Deve retornar um valor inteiro e receber um parâmetro inteiro. O método **verificarDisponibilidade**() deverá retornar 1 caso existam produtos disponíveis ou 0 em caso contrário. A existência de produtos disponíveis significa que o atributo tem quantidade maior que 0 e maior ou igual ao parâmetro recebido.
removerProdutos()	O método **removerProdutos**() retorna um inteiro e deverá receber como parâmetro a quantidade de elementos que serão removidos. Antes da remoção deve-se verificar se há disponibilidade do produto solicitado. Para isso, executar o método **verificarDisponibilidade**() e, caso este retorne 1, o método remover estoque poderá diminuir o valor recebido como parâmetro do total do atributo quantidade. O método **removerProdutos**() deverá retornar 1, caso tenha sucesso na remoção dos produtos. Caso contrário, retornar 0 informando que não foi possível remover a quantidade solicitada.

Exercício 12
Desenvolver uma classe PrincipalEstoque com a seguinte estrutura:

Métodos:

Método	**Descrição**
main()	Implementá-lo conforme o padrão da linguagem Java. Criar um objeto da classe **Estoque** e atribuir valores aos parâmetros.

Criar três objetos da classe Estoque e atribuir valores para os atributos. Exercitar a chamada dos métodos para que seja possível analisar todas as possibilidades que os métodos criados retornam.

Elementos Básicos da linguagem Java

Exercício 13
Desenvolver uma classe Java chamada ContaCorrente com a seguinte estrutura:

Atributos: conta, agencia, saldo e nome do cliente

Métodos:

Método	**Descrição**
sacar()	Retorna o valor 1 caso o saque seja realizado ou 0 se não houver saldo suficiente na conta. Deverá receber como parâmetro o valor a ser sacado.
depositar()	Realizar o depósito do valor recebido como parâmetro. Não deve retornar valor.

imprimir()	Exibir na tela os atributos da classe. Esse método não retorna nada.

Exercício 14

Desenvolver uma classe Java chamada PrincipalContaCorrente com a seguinte estrutura:

- Criar um atributo constante chamado TAM como o valor de 3.
- Criar um atributo da classe ContaCorrente do tipo vetor. Usar o comando: *ContaCorrente cc[ ] = new ContaCorrente[TAM]*. Este será usado pelos métodos da classe para realizar saques, cadastros, consultas e depósitos.
- Criar o atributo índice do tipo estático usando o seguinte comando: *public static int índice = 0;*

OBS: atenção, pois ao executar o programa só poderemos fazer um saque se já tivermos realizado um depósito e tivermos pelos menos uma conta já cadastrada.

Métodos:

Método	**Descrição**
main()	Implementá-lo conforme o padrão da linguagem Java. O método main() deverá criar um loop para o usuário escolher entre as opções cadastrar, sacar, depositar ou consultar. Se for selecionada a opção saque, executar o método execSaque(). Se selecionarmos depósito, executar o método execDeposito(). Se selecionada a opção consultar, executar o método execConsulta(). Se selecionada a opção cadastrar executar o método execCadastro().
execSaque()	Verificar se alguma conta já foi criada, realizando um teste com o atributo índice, em seguida solicitar ao usuário que escolha a posição que deseja ler. Finalmente, solicitar ao usuário que digite um valor e executar o método saca() da classe ContaCorrente usando o atributo criado. Testar o retorno do método sacar() e mostrar na tela o resultado obtido pelo valor retornado. Se for retornado 1 mostrar saque realizado, caso contrário, saque não-realizado.
execDeposito()	Verificar se alguma conta já foi criada, realizando um teste com o atributo índice, em seguida solicitar ao usuário que escolha a posição que deseja ler. Finalmente solicitar ao usuário que digite um valor e executar o método depositar() da classe ContaCorrente usando o objeto criado anteriormente.
execConsulta()	Verificar se alguma conta já foi criada, realizando um teste com o atributo índice. Em seguida, solicitar ao usuário que escolha a posição que deseja ler. Finalmente apresentar os atributos na tela executando o método imprimir() da classe ContaCorrente.
execCadastrar()	Solicitar que o usuário realize a leitura dos dados via teclado e em seguida realize a atribuição desses valores aos atributos do objeto da classe ContaCorrente, criado como atributo dessa classe. Como estamos usando vetores neste exercício, precisamos criar um novo objeto para cada novo cadastro realizado. Para isso, usar o comando: *this.cc[índice] = new ContaCorrente()*. Realizar a leitura usando o comando: *this.cc[índice],conta = sc.nextInt()*. Ao final da leitura dos dados, verificar se o atributo “indice” alcançou deu limite. Caso isso ocorra, exibir uma mensagem na tela e reinicializá-lo com 0;

Exercícios sobre o uso de construtores

Exercício 15
Desenvolver o seguinte conjunto de classes:

Classe	Descrição
Pessoa	Esta classe deve criar os atributos cpf, nome e dataNasc como privados, Na classe Pessoa criar um construtor vazio e um segundo construtor que receba três parâmetros e atribua-os a seus atributos. Não esquecer de criar todos os métodos de acesso e m´todos modificadores.
PrincipalPessoa	Esta classe deve criar um atributo que seja uma referência para o tipo da classe Pessoa. O objeto será criado (execução do operador *new*) em outro momento do exercício.

Implementar o método main() apresentando um menu para que o usuário opte por cadastrar ou imprimir os dados, no método que realizará o cadastramento, solicitar os dados de uma Pessoa e por meio do construtor atribuí-los ao objeto criado. No momento da leitura da data, realizar sua validação, a fim de garantir que seja lido no formato AAAA/MM/DD. Caso não seja lido dessa forma, realizar uma nova solicitação de leitura.
Implementar o método imprimir() que deverá exibir os dados cadastrados na tela.

Exercício 16

Desenvolver a classe ContaCorrente com os atributos: conta, agencia, saldo nomeCliente.

Implementar nessa classe o método sacar(), que retorna um inteiro informando se o saque ocorreu com sucesso e ainda recebe um parâmetro do tipo double contendo o valor do saque. Executar nesse método a lógica para realizar um saque na conta corrente. Caso o saque possa ser realizado, retornar 1. Caso contrário, retornar 0. Implementar também o método depositar(). Esse método não retorna nada e recebe como parâmetro um valor double, usado para realizar o depósito na conta corrente. Implementar o método imprimir(), que deverá exibir os valores dos atributos na tela.

Criar também um construtor com parâmetros. O construtor deverá receber os valores dos atributos da conta corrente e realizar a inicialização dos atributos com os valores recebidos. Não esquecer de criar os métodos de acesso para a classe ContaCorrente.

Desenvolver a classe PrincipalContaCorrente com um atributo como apenas uma referência para a classe ContaCorrente. A execução do operador new para a referência criada ocorrerá em um momento oportuno. Na classe PrincipalContaCorrente, criar o método main() exibindo um menu com a opção de cadastro de uma conta corrente e uma segunda opção para a impressão dos valores cadastrados. Para a opção cadastrar, criar o método cadastrar(), que deverá solicitar em variáveis locais os valores de uma conta corrente. Ao final da leitura executar o operador new para referência criada como atributo passando por meio do construtor os valores lidos.

Para a opção imprimir, criar o método imprimir() que, por meio da referência criada, exibirá os dados da conta corrente previamente cadastrada.

Exercícios com as classes Vector, String, StringBuffer e Genéricos

Exercício 17

Desenvolver uma classe Java chamada ExemploExercicioDadosPessoais contendo os seguintes atributos privados: nome, telefone e idade. Criar os métodos de acesso e métodos modificadores para os atributos (setXXX() e getXXX()).

Validar o atributo idade por meio do método setIdade(int). Caso a idade recebida seja menor que 0 ou maior que 110, apresentar uma mensagem de erro. Caso contrário, realizar a atribuição.

Criar uma segunda classe chamada ExemploExercicioVectorGenericos contendo o seguinte conteúdo: um atributo privado chamado dados do tipo Vector. Um segundo atributo chamado dadosGenericos do tipo Vector, mas definido seguindo o conceito de genérico. Esse atributo deve ser genérico para o tipo da classe ExemploExercicioDadosPessoais.

Em seguida, criar os seguintes métodos: o método main() deverá apresentar um menu com três opções: a primeira para cadastrar os dados, a segunda para listar os dados usando o atributo do tipo Vector e a terceira para listar os dados usando o atributo do tipo Vector especial para o tipo genérico da classe ExemploExercicioDadosPessoais.

A primeira opção deverá executar o método cadastrar(), a segunda, executar o método listar(), e a terceira opção deverá executar o método listarFormatoGenérico().

O método cadastrar() deverá solicitar os dados para leitura do teclado. Validar a idade lida. Caso esta seja inválida, solicitar novamente a leitura. Atribuir os dados lidos do teclado aos atributos da classe ExemploExercicioDadosPessoais. Em seguida, adicionar o objeto nos atributos dados e dadosGenericos por meio do método add() da classe Vector.

O método listar() deverá exibir na tela os valores cadastrados no atributo dados com o método get() da classe Vector. Como esse atributo não foi criado com o formato de acordo com o conceito de genérico, usar o downcasting para recuperar o objeto.

O método listaFormatoGenerico() deverá exibir os dados cadastrados no atributo dadosGenericos. Como esse atributo foi criado em acordo com o conceito de genérico, não será necessário no método get() realizar downcasting para exibir os dados na tela.

Exercício 18

Desenvolver uma classe Java chamada ExemploCarro com os seguintes atributos privados: *private int chassi, private String marca, private String fabricante, private String dtFabricacao*. Criar os métodos de acesso e métodos modificadores para os atributos.

Criar uma segunda classe chamada ExemploCarroPrincipal contendo os seguinte conteúdo: um atributo chamado carro do tipo Vector, mas definido segundo o conceito de genérico. Esse atributo deve ser genérico para o tipo da classe ExemploCarro. Em seguida, criar os seguintes métodos:

Método	Descrição
main()	Deverá exibir um menu com duas opções: a primeira para cadastrar os dados e a segunda para lista-los. A primeira opção deverá executar o método cadastrar(), e a segunda, o método imprimir().

cadastrar()	Deverá solicitar os dados para leitura do teclado. Atribuir os dados lidos do teclado aos atributos da classe ExemploCarro. Em seguida, adicionar o objeto do tipo da classe ExemploCarro no atributo carro do tipo Vector.
imprimir()	Deverá exibir na tela os valores cadastrados no atributo carro usando o método get() da classe Vector. Esse método deverá apresentar a data de fabricação lida no formato AAAA/MM/DD no formato DD/MM/AAAA. Para isso, usar o método Split() da classe String.

Exercício usando herança e classe abstrata

Exercício 19

Desenvolver o seguinte conjunto de classes.

- **Primeira classe**: classe chamada VetorUnidimensional.

  Deve conter como atributos:

Atributo	Descrição
dim1	Representa um vetor de inteiros. Usar o formato de vetor com colchetes. Ou seja, neste exercício não devemos usar a classe Vector. O objetivo é praticar o conceito de herança, e para termos um array e uma matriz precisamos usar o formato tradicional.
linha	Deve ser protegido e inteiro.
TAMANHO	Representa um atributo constante com valor igual a 2. Definir com o modificador final e protegido.

Deve conter os seguintes construtores:

- Sem argumento: criar um objeto para o atributo dim1 usando como tamanho do vetor o atributo constante TAMANHO.
- Com um argumento inteiro: deve verificar se o parâmetro recebido viola o tamanho do inteiro (+-2.000.0000) ou é negativo. Caso viole mostrar uma mensagem de erro. Caso contrário, criar um vetor com o tamanho recebido.

Deve conter os seguintes métodos:

Método	Descrição
adicionar()	Este método recebe um parâmetro que será o valor a ser inserido no vetor e não retorna nada. Criar a lógica para garantir que não seja excedido o vetor na inclusão de novos valores. **Dica**: O vetor estará excedido quando linha for igual ao tamanho do vetor.
imprimir()	Apresentar o vetor na tela.

- **Segunda classe**: classe chamada MatrizBidimensional que deve estender a classe VetorUnidimensional.

  A classe MatrizBidimensional deve conter como atributos:

Atributo	Descrição
dim2	Representa uma matriz de inteiros.
coluna	Deve ser privado e inteiro.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar a matriz usando como tamanho o atributo TAMANHO.
- Com dois argumentos inteiros: deve verificar se os parâmetros recebidos violam o tamanho do inteiro (+-2.000.000) ou são negativos. Caso violem, mostrar uma mensagem de erro. Caso contrário, criar a matriz com os tamanhos recebidos.

Deve conter os seguintes métodos:

Método	Descrição
adicionar()	Recebe um parâmetro que será o valor a ser inserido na matriz. Criar a lógica para garantir que não seja excedida a matriz na inclusão de novos valores. **Dica**: caso a linha seja menor que o tamanho da primeira dimensão, incrementar a coluna. Quando a coluna atingir o máximo, incrementar a linha e zerar a coluna. Quando a linha e a coluna alcançarem o limite, zerar linha e coluna.
imprimir()	Apresentar a matriz na tela.

- Classe chamada **Principal**.

Deve conter como atributos:

Atributo	Descrição
dimensao	Um objeto do tipo da classe VetorUnidimensional

Deve conter os métodos:

- main() deve executar o método executar().
- executar(): representar um menu para que seja escolhido entre vetor e matriz.

Caso selecionado vetor, apresentar outro menu para que escolha entre as seguintes opções:

- Adicionar em um vetor com tamanho-padrão (TAMANHO).
- Adicionar em um vetor com tamanho definido por um valor lido do teclado.
- Imprimir o vetor. Esse método deverá imprimir os elementos do vetor criado.

Caso selecionada a matriz, apresentar outro menu para a escolha entre:

- Adicionar em uma matriz com tamanho-padrão (TAMANHO).
- Adicionar em uma matriz com tamanho, sendo definido por dois valores lidos do teclado ou ainda imprimir a matriz.

OBS: os métodos na classe Principal que realizarem a criação de um vetor ou matriz com tamanho determinado pelo usuário deverão validar se o valor passado como parâmetro está inválido. Se estiver, esses métodos devem solicitar que a leitura seja novamente realizada.

Exercício 20

Desenvolver o seguinte conjunto de classes.

- **Primeira classe**: classe abstrata chamada ClasseAbstrataDimensao. Deve definir os seguintes métodos abstratos.
    - imprimir() e adicionar(). O método adicionar() deve receber como parâmetro um valor inteiro. Ambos não retornam nada.
    - A classe abstrata também deve definir um atributo constante chamado TAMANHO com valor igual a 2.
- **Segunda classe**: classe chamada VetorUnidimensional que deve estender a classe abstrata apresentada.

Deve conter como atributos protegidos:

Atributo	**Descrição**
dim1	Representa um vetor de inteiros. Usar o formato de vetor com colchetes.
linha	Representa as linhas do vetor.

Deve conter os seguintes construtores:

- Sem argumento: criar um objeto para o atributo dim1 usando como tamanho do vetor o atributo constante TAMANHO.
- Com um argumento inteiro: deve verificar se os parâmetros recebidos violam o tamanho do inteiro (+-2.000.000) ou é negativo. Caso viole, mostrar uma mensagem de erro. Caso contrário, criar o vetor com o tamanho recebido.

Deve conter os seguintes métodos:

Método	**Descrição**
adicionar()	Recebe um parâmetro que será o valor a ser inserido no vetor e não retorna nada. Criar a lógica para garantir que não seja excedido o vetor na inclusão de novos valores. **Dica**: O vetor estará excedido quando linha for igual ao tamanho do vetor.
imprimir()	Apresentar o vetor na tela.

OBS: a implementação dos métodos abstratos na subclasse é necessária para não haver erros de compilação. Um método abstrato deve ser implementado pelas classes concretas.

- **Terceira classe**: classe chamada MatrizBidimensional que deve estender a classe VetorUnidimensional.

Deve conter como atributos protegidos:

Atributo	**Descrição**
dim2	Representa uma matriz de inteiros. **OBS**: matriz contém mais de uma dimensão.
coluna	Deve ser privado.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar a matriz usando como tamanho o atributo TAMANHO.
- Com dois argumentos inteiros: deve verificar se os parâmetros recebidos violam o tamanho do inteiro (+-2.000.000) ou são negativos. Caso violem, mostrar uma mensagem de erro. Caso contrário, criar a matriz com os tamanhos recebidos.

Deve conter os seguintes métodos:

Método	**Descrição**
adicionar()	Recebe um parâmetro que será o valor a ser inserido na matriz. Criar a lógica para garantir que não seja excedida a matriz na inclusão de novos valores. **Dica**: caso a linha seja menor que o tamanho da primeira dimensão, incrementar a coluna. Quando a coluna atingir o máximo, incrementar a linha e zerar a coluna. Quando a linha e a coluna alcançarem o limite, zerar linha e coluna.
imprimir()	Apresentar a matriz na tela.

- Classe chamada **Principal**.

Deve conter como atributos:

Atributo	**Descrição**
dimensao	Um objeto do tipo da classe ClasseAbstrataDimensao

Deve conter o método:

- executar(): representar um menu para que seja escolhido entre vetor e matriz.

Caso selecionado vetor, apresentar outro menu para que escolha entre as seguintes opções:

- Adicionar em um vetor com tamanho-padrão (TAMANHO).
- Adicionar em um vetor com tamanho definido por um valor lido do teclado.
- Imprimir o vetor. Esse método deverá imprimir os elementos do vetor criado.

Caso selecionada a matriz, apresentar outro menu para a escolha entre:

- Adicionar em uma matriz com tamanho-padrão (TAMANHO).
- Adicionar em uma matriz com tamanho, sendo definido por dois valores lidos do teclado ou ainda imprimir a matriz.
- Imprimir a matriz. Esse método deverá imprimir os elementos da matriz criada.

Método main() deve executar o método executar().

OBS: os métodos na classe Principal que realizarem a criação de um vetor ou matriz com tamanho determinado pelo usuário deverão validar se o valor passado como parâmetro está inválido. Se estiver, esses métodos devem solicitar que a leitura seja novamente realizada.

Exercício 21

Desenvolver o seguinte conjunto de classes:

- Classe Pessoa
  - Criar a classe Pessoa como abstrata e com o atributo privado nome e seus métodos de acesso. Criar o método imprimir() como abstrato. Criar um construtor que deverá inicializar o atributo da classe.

- Classe PessoaFisica
  - Criar a classe PessoaFisica que deverá ser uma subclasse de Pessoa. Criar os atributos privados cpf e data de nascimento. Criar os métodos de acesso.
  - Criar para a classe PessoaFisica um construtor vazio e um segundo construtor que receba os parâmetros da classe e o parâmetro de sua superclasse. Inicializar os atributos da subclasse e chamar o comando **super** com o parâmetro da superclasse.
  - Desenvolver o método imprimir() que apresenta os valores na tela.

- Classe PessoaJuridica
  - Criar a classe PessoaJuridica que também estende da classe Pessoa. Criar os atributos cnpj, inscrição estadual, nome fantasia e razão social. Criar métodos de acesso. Desenvolver o método imprimir(), que exibirá os dados na tela.
  - Criar para a classe PessoaJuridica um construtor vazio e um segundo construtor que receba os parâmetros da classe e o parâmetro da sua superclasse. Inicializar os atributos da classe e chamar o comando **super** com o parâmetro da superclasse.

- Classe PrincipalPessoa
  - Desenvolver a classe PrincipalPessoa com o atributo pessoa do tipo da classe Pessoa. Não será necessário criar objeto neste momento, ou seja, iremos faze-lo depois que o usuário escolher uma opção.
  - Implementar o método main() e apresentar ao usuário um menu com as opções cadastrar e imprimir. Se o usuário escolher cadastrar, executar o método cadastrar(). Se o usuário optar por imprimir, executar o método imprimir().
  - O método imprimir() deverá verificar se o atributo da classe está igual a nulo e, em caso afirmativo, exibir uma mensagem de erro na tela. Caso contrário deverá executar o método imprimir() usando o atributo pessoa para isso. Não esquecer de verificar se o atributo pessoa está nulo. Caso esteja, apresentar uma mensagem de erro. Criaremos um objeto par o atributo pessoa no método cadastrar().
  - No método cadastrar() deve ser solicitado que o usuário escolha entre pessoa jurídica ou física. Se escolher pessoa física, solicitar que o usuário leia com o teclado em variáveis locais os valores referentes a uma pessoa física. Ao concluir, criar um objeto do tipo da classe PessoaFisica usando o construtor com parâmetros. Realizar o mesmo procedimento quando o usuário escolher a opção de cadastrar uma pessoa jurídica.

Exercício com uso de interface

Exercício 22

Desenvolver o seguinte conjunto de classes e interfaces:

- Desenvolver a interface InterfaceDimensao com os seguintes métodos:
  - imprimir() e adicionar(). Ambos não retornam nada. O método adicionar() deve receber como parâmetro um valor inteiro.

- Incluir também na InterfaceDimensao um atributo chamado TAMANHO com valor igual a 2.

- Criar uma classe chamada VetorUnidimensional que deve implementar a interface InterfaceDimensao. Deve conter como atributos:

Atributo	**Descrição**
dim1	Representa um vetor de inteiros. Criar como privado.
linha	Deve ser protegido e do tipo inteiro.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar o vetor usando como tamanho o atributo da interface apresentada.
- Com um argumento inteiro: deve verificar se o parâmetro recebido é maior que 2.000.000 ou menor ou igual a zero. Caso viole essa determinação, exibir uma mensagem de erro. Caso contrário, criar o vetor com o tamanho recebido.

Deve conter os seguintes métodos:

Método	**Descrição**
adicionar()	Recebe um parâmetro que será o valor a ser inserido no vetor. Criar a lógica para garantir que não seja excedido o vetor na inclusão de novos valores. **Dica**: O vetor estará excedido quando linha for igual ao tamanho do vetor.
imprimir()	Apresentar o vetor na tela.

- Desenvolver uma segunda classe chamada MatrizBidimensional que deve implementar a interface apresentada e estender a classe VetorUnidimensional. Deve conter como atributos:

Atributo	**Descrição**
dim2	Representa uma matriz de inteiros.
coluna	Deve ser privado e inteiro.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar a matriz usando como tamanho da interface apresentada.
- Com dois argumentos inteiros: deve verificar se o parâmetro recebido é maior que 2.000.000 ou menor ou igual a zero. Caso viole essa determinação, exibir uma mensagem de erro. Caso contrário, criar a matriz com o tamanho recebido.

Deve conter os seguintes métodos:

Método	**Descrição**
adicionar()	Recebe um parâmetro que será o valor a ser inserido na matriz. Criar a lógica para garantir que não seja excedida a matriz na inclusão de novos valores. **Dica**: caso a linha seja menor que o tamanho da primeira dimensão, incrementar a coluna. Quando a coluna atingir o máximo, incrementar a linha e zerar a coluna. Quando a linha e a coluna alcançarem o limite, zerar linha e coluna.
imprimir()	Apresentar a matriz na tela.

- Desenvolver uma classe chamada **Principal**. Deve conter como atributos:

Atributo	Descrição
dimensao	Deve ser uma referência para o tipo de interface criada.

Deve conter os métodos:

Método	Descrição
main()	Deve executar o método executar()
executar()	Representar um menu para a escolha entre vetor e matriz. Caso seja selecionado vetor, apresentar outro menu para a escolha entre as seguintes opções:

- Adicionar em um vetor com tamanho-padrão (TAMANHO). caso seja selecionada essa opção, executar o método vetorAdicionarPadrao(). O método deverá criar um objeto do tipo VetorUnidimensional usando o construtor sem parâmetros. Também deve executar em seguida o método adicionar() presente na classe Principal.

- Adicionar em um vetor com tamanho definido por um valor lido do teclado. Se escolhida essa opção, executar o método vetorAdicionarTamEspecifico(). O método deverá solicitar que o usuário digite um tamanho para o vetor. Em seguida, criar o objeto do tipo VetorUnidimensional usando o construtor, que recebe um parâmetro lido pelo usuário. Esse método deve executar em seguida o método adicionar() presente na classe Principal.

- Imprimir o vetor. Esse método deverá imprimir os elementos do vetor criado.

Caso selecionada a matriz, apresentar outro menu para a escolha entre as seguintes opções:

- Adicionar em uma matriz com tamanho-padrão (TAMANHO).

- Adicionar em uma matriz com tamanho, sendo definido por dois valores lidos do teclado.

- Imprimir a matriz. Esse método deverá imprimir os elementos da matriz criada.

Após a definição de como a matriz ou o vetor será criado, executar o método adicionar() do vetor da matriz, dependendo da opção escolhida. Como ambas as classes implementam a interface por meio do conceito de polimorfismo, poderemos realizar uma única chamada ao método adicionar(). As classes VetorUnidimensional e MatrizBidimensional, por implementarem a interface InterfaceDimensao, tem o método adicionar().

```
//fragmento de código que representa um exemplo do método adicionar
private void adicionar(){
	int valor = 0;
	Scanner sc = new Scanner(System.in);
	while(true){

	System.out.println("*********************************************************");
		System.out.println("Entre com o valor para ser inserido < 0 - para
finalizar>: ");
		valor = sc.nextInt();
		if(valor == 0){
			break;
		}
		//aqui estamos fazendo uso de polimorfismo
		this.dimensao.adicionar(valor);
	}
}
```

Exercício com interface e exceções

Exercício 23

Desenvolver o seguinte conjunto de classes e interfaces:

- Desenvolver a interface InterfaceDimensao com os seguintes métodos:
    - imprimir() e adicionar(). Ambos não retornam nada, o método adicionar() deve receber como parâmetro um valor inteiro. Esse método deve relançar uma exceção do tipo MyClassException. Ambos os métodos não retornam nada.
    - Incluir também um atributo chamado TAMANHO com valor 2.
- Desenvolver uma classe chamada VetorUnidimensional , que deve implementar a interface desenvolvida. Essa classe deve conter como atributos:

Atributo	**Descrição**
dim1	Representa um vetor de inteiros. Criar como privado e no formato array.
linha	Deve ser protegido e do tipo inteiro.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar o vetor usando como tamanho o atributo da interface apresentada.
- Com um argumento inteiro: deve verificar se o parâmetro recebido é maior que 2.000.000 ou menor ou igual a zero. Caso viole essa determinação, lançar uma exceção do tipo MyClasseException com uma mensagem de erro. Caso contrário, criar o vetor com o tamanho recebido.

Deve conter os seguintes métodos:

Método	**Descrição**
adicionar()	Recebe um parâmetro que será o valor a ser inserido no vetor. Criar a lógica para garantir que não seja excedido o vetor na inclusão de novos valores. Caso o vetor exceda a quantidade, lançar uma exceção avisando sobre a possível sobreposição de valores do vetor. **Dica**: O vetor estará excedido quando linha for igual ao tamanho do vetor.
imprimir()	Apresentar o vetor na tela.

- Desenvolver uma segunda classe chamada MatrizBidimensional que deve implementar a interface apresentada e estender a classe VetorUnidimensional. Deve conter como atributos:

Atributo	**Descrição**
dim2	Representa uma matriz de inteiros. Usar o padrão de array para criar essa matriz. Deve ser privada.
coluna	Deve ser privado e inteiro.

Deve conter os seguintes construtores:

- Sem argumento: deverá criar a matriz usando como tamanho da interface apresentada.
- Com dois argumentos inteiros: deve verificar se o parâmetro recebido é maior que 2.000.000 ou menor ou igual a zero. Caso viole essa determinação, lançar uma exceção do tipo MyClasseException com uma mensagem de erro. Caso contrário, criar a matriz com o tamanho recebido.

Deve conter os seguintes métodos:

Método	Descrição
adicionar()	Recebe um parâmetro que será o valor a ser inserido na matriz. Criar a lógica para garantir que não seja excedida a matriz na inclusão de novos valores. Caso a matriz exceda a quantidade, lançar uma exceção avisando sobre a possível sobreposição de valores na matriz. **Dica**: caso a linha seja menor que o tamanho da primeira dimensão, incrementar a coluna. Quando a coluna atingir o máximo, incrementar a linha e zerar a coluna. Quando a linha e a coluna alcançarem o limite, zerar linha e coluna da matriz.
imprimir()	Apresentar a matriz na tela.

- Desenvolver uma classe chamada **Principal**. Deve conter como atributos:

Atributo	Descrição
dimensao	Deve ser uma referência para o tipo de interface criada.

Deve conter os métodos:

Método	Descrição
main()	Deve executar o método executar()
executar()	Representar um menu para a escolha entre vetor e matriz.

Caso seja selecionado vetor, apresentar outro menu para a escolha entres seguintes opções:

- Adicionar em um vetor com tamanho-padrão (TAMANHO). caso seja selecionada essa opção, executar o método vetorAdicionarPadrao(). O método deverá criar um objeto do tipo VetorUnidimensional usando o construtor sem parâmetros. Também deve executar em seguida o método adicionar() presente na classe Principal.

- Adicionar em um vetor com tamanho definido por um valor lido do teclado. Se escolhida essa opção, executar o método vetorAdicionarTamEspecifico(). O método deverá solicitar que o usuário digite um tamanho para o vetor. Em seguida, criar o objeto do tipo VetorUnidimensional usando o construtor, que recebe um parâmetro lido pelo usuário. O método deverá capturar a exceção caso seja digitado um valor que exceda os limites do vetor. Ainda no método vetorAdicionarTamEspecifico(), devemos executar em seguida o método adicionar() presente na classe Principal.

O fragmento de código abaixo mostra um exemplo a ser usado no método para a captura da exceção:

```
//fragmento de código para o método executar
      try{
            dimensão = new VetorUnidimensional(tam);
            adicionar();
      }catch(MyClassException e){
            eSystem.out.print(e.getMessage();
      }
```

EXERCÍCICIOS COMPLEMENTARES

1. Explique a diferença entre:
   a. Um objeto e uma referência a objeto.
   b. Um objeto e uma variável de objeto.
   c. Um objeto e uma classe.
   d. Um construtor e um método
   e. Uma variável de instância e uma variável local
   f. Uma variável local e uma variável de parâmetro
   g. `new BankAcount(5000);` e `BankAcount b = new BankAcount(5000);`
   h. `BankAcount b;` e `BankAcount b = new BankAcount(5000);`

2. Encontre os erros nas seguintes instruções:

```
Rectangle r = (5,10,15,20);

double x = BankAcount(10000).getBalance();

BankAcount b;
d.deposit(10000);

b = new BanckAcount(10000);
b.add("one million bucks");
```

3. Implemente uma classe `Funcionário`. Um empregado tem um nome (um `String`) e um salário (um `double`). Escreva um construtor *default*, um construtor com dois parâmetros (nome e salário) e métodos para devolver nome e salário. Escreva um pequeno programa que teste a sua classe.

4. Aprimore a classe do exercício anterior para adicionar um método `aumentarSalario(doublé byPercent)` que aumente o salário do funcionário em uma certa porcentagem. Exemplo de uso:

```
Funcionário harry = new Funcionário("Hacker, Harry", 55000);

Harry.aumentarSalrio(10); // Harry consegue um aumento de 10%
```

5. Implemente uma classe `Carro` com as seguintes propriedades: um veículo tem um certo consumo de combustível (medidos em milhas/galão ou litros/km – escolha um) e uma certa quantidade de combustível no tanque. O consumo é especificado no construtor e o nível de combustível inicial é 0. Forneça um método `andar` que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina, e os métodos `obterGasolina`, que devolvam o nível atual de combustível, e `adicionarGasolina`, para abastecer o tanque. Exemplo de uso:

```
Carro myFerrari = new Carro(29);// 29 litros/km

myFerrari.adicionarGasolina(20);// abastece com 20 litros

myFerrari.andar(100);              //anda 100 km
System.out.println(myFerrari.obterGasolina());  // imprime o
combustível que resta
```

6. Implemente uma classe `estudante`. Para o objetivo deste exercício, um aluno tem um nome e uma contagem total de provas. Forneça um construtor apropriado e os métodos `obterNome()`, `obterProva(int nota)`, `obterNotaTotal()` e `obterMedia()`. Para calcular a última, você também precisa armazenar o número de provas que o aluno fez.

7. Implemente a classe `Produto`. Um produto tem um nome e um preço, por exemplo `new Produto("Torradeira", 29.95)`. Forneça métodos `obterNome()`, `obterPreço()` e `estabelecerPreco()`. Escreva um programa que crie dois produtos, imprima o nome e o preço, reduza seus preços em R$ 5,00 e depois os imprima novamente.

8. Implemente uma classe `Circulo` que tenha métodos `obterArea()` e `obterPerimetro()`. No construtor, forneça o raio do círculo.

9. Implemente uma classe `Quadrado` que tenha métodos `obterArea()` e `obterPerimetro()`. No construtor, forneça a largura do quadrado.

10. Implemente uma classe `LataDeRefrigerante` com os métodos `ObterAreaSuperficie()` e `obterVolume()`, forneça a altura e o raio da lata.

11. Implemente uma classe `PopulacaoBaratas` que simule o crescimento de uma população de baratas. O construtor recebe o tamanho da população inicial das baratas. O método `WaitForDouble` simula um período durante o qual a população dobra. O método `spray` pulveriza as baratas com inseticida. O que reduz a população em 10%. O método `obterBaratas` devolve o número atual de baratas. Implemente a classe e um programa de teste que simule uma cozinha que começa com 10 baratas. Espere, use o inseticida `spray` e, imprima a contagem de baratas. Repita a operação três vezes.

12. Implemente uma classe `PopulacaoCoelhos` que simule o crescimento de uma população de coelhos. As regras são as seguintes: comece com um casal de coelhos. Os coelhos são capazes de acasalar na idade de um mês. Um mês mais tarde, cada fêmea produz outro casal de coelhos. Parta do princípio de que os coelhos nunca morrem e que a fêmea sempre produz um novo casal (um macho, uma fêmea) a cada mês, a partir do segundo mês. Implemente um método `WaitAmonth` que espere um mês e um método `obterCasais` que imprima o número atual de casais de coelho. Escreva um programa de teste que mostre o crescimento da população de coelhos em 10 meses. **DICA**: mantenha um campo de instância para os casais recém-nascidos de coelhos e outro para os casais de coelhos que têm, pelo menos, um mês de idade.

Projetando classes

Assuntos abordados

- Coesão e acoplamento
- Métodos de acesso e métodos modificadores
- Métodos estáticos e campos estáticos
- Escopo
- Pacotes

Questões

13. Considere a descrição dos problemas a seguir:
    a. Os usuários colocam moedas em uma maquia automática de venda e selecionam um produto pressionando um botão. Se as moedas inseridas forem suficientes para cobrir o preço de compra do produto, este será entregue e o troco, fornecido. Caso contrário, as moedas inseridas são devolvidas ao usuário. Quais classes você deve utilizar para implementar esse problema?
    b. Os funcionários recebem seu contra-cheque quinzenalmente. Eles são pagos pelo número de horas trabalhadas; entretanto, se trabalharam mais e 40 horas por semana, recebem horas extras a 150% dos seus salários normais. Quais classes você deve utilizar para implementar esse problema?

c. Os clientes fazem pedidos de produtos a uma loja. A loja emite faturas para listar os itens e as quantidades solicitadas, os pagamentos recebidos e os valores devidos. Os produtos são despachados para o endereço de remessa do cliente e as faturas são enviadas ao endereço de cobrança. Quais classes você deve utilizar para implementar esse problema?

14. Suponha que um objeto Invoice (Fatura) contenha descrições dos produtos pedidos e o endereço de cobrança e remessa do cliente. Desenhe um diagrama UML mostrando as dependências entre as classes Invoice (Fatura), Address (Endereco), Customer (Cliente) e Product (Produto).

15. Considere uma máquina automática de venda de produtos e que os usuários coloquem moedas nela para comprar esses produtos. Desenhe um diagrama UML mostrando as dependências entre as classes VendingMachine (MaquinaDeVenda), Coin (Moeda) e Product (Produto).

16. Na classe seguinte, a variável **n** ocorre em vários escopos. Quais declarações de **n** são válidas e quais são ilegais?

```
public class X{
      public int f(){
            int n = 1;
            retun n;
      }

      public int g(int k){
            int a;
            for(int n = 1; n <= k;n++)
                  a = a + n;
            return a;
      }

      public int h(int n){
            int b;
            for(int n = 1;n <= 10; n++)
                  b = b + n;
            return b + n;
      }

      public int k(int n){
            if (n < 0){
                  int k = -n;
                  int n = (int)(Math.sqrt(n));
                  return n;
            }else
                  return n;
      }

      public int n(int k){
            int a;
            for(int n = 1; n <= k; n++)
                  a = a + n;
            for(int n = k; n >= 1; n++)
                  a = a + n;
            return a;
      }
      private int n;
}
```

Herança

- Hierarquia de classes
- Herança de métodos e de campos de instância
- Construção de sub-classes
- Conversão de sub-classes em superclasses
- Controle de acesso

Questões

17. Nos pares de classes a seguir, identifique a superclasse e a subclasse:
    - Funcionário, Gerente
    - Polígono, Triangulo
    - AlunoDePosGraduacao, Aluno
    - Pessoa, Estudante
    - Funcionário, AlunoDePosGraduacao
    - ContaBancaria, ContaCorrente
    - Veiculo, Carro
    - Veículo, Minivan
    - Carro, Minivan
    - Caminhao, Veiculo

18. Suponha que a classe `sub` estenda a classe `Sanduíche`. Quais das atribuições são válidas?

```
Sanduíche x = new Sanduíche();
Sub y = new sub();
x = y;
y = x;
y = new Sanduiche();
x = new sub();
```

19. Desenhe um diagrama de herança que mostre as relações de herança entre as classes:
    - Pessoa
    - Funcionário
    - Aluno
    - Instrutor
    - SalaDeAula
    - Objeto

20. Em um sistema orientado a objeto para simulação de tráfego, temos as seguintes classes:
    - Veículo
    - Carro
    - Caminhão
    - Sedan
    - Conversível
    - CaminhaoPicape
    - VeiculoEsporte
    - Minivan
    - Bicicleta
    - Motocicleta

    Desenhe um diagrama de herança que exiba as relações entre as classes.

21. Quais as relações de herança você estabeleceria entre as seguintes classes?
    - Aluno

- Professor
- ProfessorAssistente
- Funcionário
- Secretaria
- ChefeDeDepartamento
- Zelador
- PalestranteEmSeminario
- Pessoa
- Curso
- Palestra
- LaboratorioDeInformatica

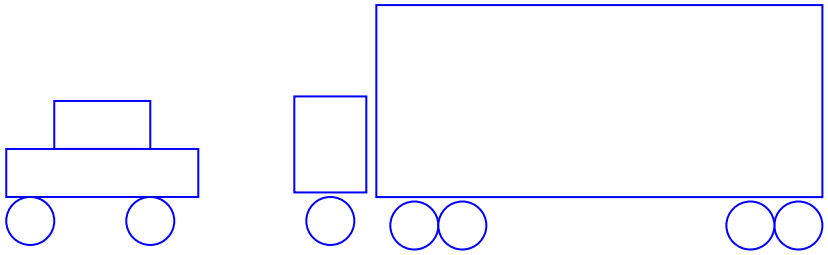
22. Explique os dois significados da palavra-chave `super`. Explique os dois significados da palavra-chave `this`. Como elas estão relacionadas?

23. Implemente uma superclasse `Pessoa`. Faça com que duas classes, `Aluno` e `Instrutor`, herdem de `Pessoa`. A pessoa tem nome e ano de nascimento. O aluno tem nota e o instrutor tem salário. Escreva as definições de classe, os construtores e os métodos `toString` para todas as classes. Forneça um programa de testes que teste essas classes e métodos.

24. Crie uma classe `Funcionario` com um nome e salário. Crie uma classe `Gerente` herdada de `Funcionário`. Adicione um campo de instância, chamado `departamento`, do tipo `String`. Forneça um método `toString` que imprime nome, departamento e salário do gerente. Crie uma classe `Executivo` herdada de `Gerente`. Forneça métodos `toString` apropriados para todas as classes. Crie um programa de teste que teste essas classes e métodos.

25. Escreva uma superclasse `Trabalhador` e subclasses `TrabalhadorPorHora` e `TrabalhadorAssalariado`. Cada trabalhador tem um nome e salário pago mensalmente. Escreva um método `CalcularPagamento (int horas)` que calcule o pagamento semanal de cada trabalhador. O trabalhador que ganha por hora é pago, obviamente, de acordo com o número real de horas trabalhadas, sendo `horas`, no máximo, igual a 40. Se ele trabalhou mais de 40 horas, cada hora excedente é paga como uma hora e meia. O trabalhador assalariado é pago pela carga horária de 40 horas, independentemente de qual seja o número real de horas trabalhadas. Crie um programa de testes que utiliza polimorfismo para testar essas classes e métodos.

26. Implemente uma superclasse `Veiculo` e subclasse `Carro` e `Caminhão`. O veículo tem uma posição na tela. Escreva métodos `draw` que desenhem os carros e caminhões da seguinte forma: escreva um método `VeiculoAleatorio` que gere referencias a `Veiculo` aleatoriamente, com uma probabilidade igual para a construção de carros e caminhões, com posições aleatórias. Chame esse método 10 vezes e desenhe todos eles.



Interfaces e Polimorfismo
Assuntos abordados:

- Desenvolvimento de soluções reutilizáveis
- Conversão de tipos
- Polimorfismo
- Uso de interfaces para aprimorar a capacidade de reutilização
- Classes internas

Questões

27. Suponha que C seja uma classe que implemente as interfaces I e J. Quais das seguintes atribuições precisão de coerção?

```
C c = . . . ;
I i = . . . ;
J j = . . . ;
c = i;     //1
j = c;     //2
i = j;     //3
```

28. Suponha que C seja uma classe que implemente as interfaces I e J. Quais das seguintes atribuições dispararão uma exceção?

```
C c = new C() ;
I i = c;         //1
J j = (J)i;      //2
C d = (C) i;     //3
```

29. Suponha que a classe `Sanduíche` implemente a interface `Comestível`. Quais das seguintes atribuições são válidas?

```
Sanduíche sub = new Sanduíche();
Comestível e = sub;              //1
Rectangle caixaDeCereal = new Rectangle(5,10,20,30);
Comestível f = caixaDeCereal;    //2
f = (Comestível)caixaDeCereal;   //3
sub = e;                         //4
sub = (Sanduíche)e;              //5
sub = (Sanduíche)caixaDeCereal; //6
```

30. As classes `Rectangle2.Double`, `Ellipse2D.Double` e `Line2D.Double` implementam a interface `Shape`. A classe `Graphics2D` depende da interface `Shape`, mas não das classes retângulo, elipse e linha. Desenhe um diagrama UML que represente esses fatos.

31. As classes `Rectangle2.Double`, `Ellipse2D.Double` e `Line2D.Double` implementam a interface `Shape`. A interface `Shape` tem um método

```
Rectangle getBounds()
```

que retorna um retângulo que delimita completamente a forma. Considere a chamada de método:

```
Shape s = . . .
Rectangle r = s.getBounds();
```

Explique por que isso é um exemplo de polimorfismo.

Tratamento de Exceções

Assuntos abordados:

- Lançamento de exceções
- Exceções verificadas

- Projeto de exceções
- Captura de exceções
- Cláusula finally

Questões

32. Qual a diferença entre lançar e capturar uma exceção?

33. O que é uma exceção verificada? O que é uma exceção não-verificada? Uma `NullPointerException` é verificada ou não-verificada? De que exceções você precisa para declarar com a palavra-chave `throws`?

34. Por que você não precisa declarar que seu método pode lançar uma `NullPointerException`?

35. Quando seu programa executa uma instrução `throw`, qual instrução é executada em seguida?

36. O que acontece se uma exceção não tem uma cláusula `catch` correspondente?

37. O que o seu programa pode fazer com o objeto `exception` que uma cláusula `catch` recebe?

38. O tipo de objeto `exception` é sempre do mesmo tipo declarado na cláusula `catch` que o captura?

39. Que tipo de objeto você pode lançar? Você pode lançar um `String`? E um inteiro?

40. Qual é o objetivo da cláusula `finally`? Dê um exemplo de como ela pode ser usada.

41. O que acontece quando é lançada uma exceção? O código de uma cláusula `finally` é executado, e ele lança uma exceção de um tipo diferente do original? Qual é capturado pela cláusula `catch` envolvente? Escreva um programa de exemplo para testar isso.