Exercícios Linguagem Java

Básico

1. Fazer um programa em Java para receber um intervalo de tempo do usuário e transformá-lo no seu equivalente em horas, minutos e segundos.

2. Implementar um gerador de números primos. O gerador recebe do usuário (via teclado) um valor inteiro *n* e calcula e mostra os *n* primeiros números primos correspondentes no console

3. A distância entre várias cidades é dada por uma tabela como a exemplificada abaixo (em km):

	1	2	3	4	5
1	00	15	30	05	12
2	15	00	10	17	28
3	30	10	00	03	11
4	05	17	03	00	80
5	12	28	11	80	00

   Implemente um programa que:
   - leia uma tabela semelhante à exemplificada em um *array* bidimensional, obtendo as distâncias relativas do usuário. O programa não deve perguntar distâncias já informadas (por exemplo, se o usuário já forneceu a distância entre 1 e 3 não é necessário informar a distância entre 3 e 1, que é a mesma) e também não deve perguntar a distância de uma cidade para ela mesma, que é 0.
   - leia um percurso fornecido pelo usuário em um *array* unidimensional. Calcule e mostre a distância percorrida. Por exemplo: dado o percurso 1, 2, 3, 2, 5, 1, 4, para a tabela mostrada como exemplo teremos: 15 + 10 + 10 + 28 + 12 + 5 = 80 km.

4. Fazer uma classe Java para:

   - receber um número N do usuário
   - receber N números inteiros em um *array* unidimensional
   - classificar em ordem crescente utilizando o método da bolha (*bubble-sort*) mostrar o *array* classificado

5. Implementar um programa para:

- Perguntar ao usuário dois valores de coordenadas geográficas, com latitude e longitude
    - Latitude: gg:mm.ddddd
    - Longitude:ggg:mm.ddddd
- Informar se as coordenadas são inválidas e pedir novamente, se for o caso
- Calcular e mostrar a distância, em metros, entre as duas coordenadas

6. Fazer uma classe ExecString que:
    a. Recebe duas strings do usuário
    b. Conta e informa quantas vezes a segunda string ocorre dentro da primeira
    c. Informa uma estatística dos caracteres contidos nas 2 strings.

7. Fazer um programa em Java para:
- receber uma string do usuário
- contar e mostrar quantas palavras existem na string
- contar e mostrar quantas vogais existem na string

8. Fazer uma classe Ex2Sorteio para:
    - Sortear um número de 0 a 1000 (dica: usar *Math.random()*)
    - Pedir um palpite ao usuário. Se ele errar, informar se o palpite é maior ou menor do que o número sorteado.
    - Pedir novos palpites até que o usuário acerte e, depois disso, mostrar em quantas tentativas ele acertou.

9. Torre de Hanói: considerando 3 torres, o objetivo é transferir 3 discos que estão na torre A para a torre C, usando uma torre B como auxiliar. Somente o último disco de cima de uma pilha pode ser deslocado para outra, e um disco maior nunca pode ser colocado sobre um menor. Implementar uma classe com um método recursivo que mostra a seqüência de movimentos para resolver o problema da Torre de Hanói.

10. Fazer um programa em Java para:
- perguntar ao usuário quantos alunos existem em uma turma
- receber a média final de cada um dos alunos da turma e armazenar em um array
- calcular e mostrar a média da turma e quantos alunos ficaram acima e abaixo desta média

Orientação a Objetos

11. Criar a classe *Pessoa* com as seguintes características:
    - atributos: idade e dia, mês e ano de nascimento, nome da pessoa
    - métodos:
        - *calculaIdade(),* que recebe a data atual em dias, mês e anos e calcula e armazena no atributo *idade* a idade atual da pessoa, sem retornar valor
        - *getIdade(),* que retorna o valor da idade
        - *getNome(),* que retorna o nome da pessoa
        - *setNome(),* que recebe o nome da pessoa como parâmetro e inicializa o atributo da classe
        - construtora que recebe nome e dia, mês e ano de nascimento como parâmetros e preenche nos atributos correspondentes do objeto.
    - Fazer uma classe principal que crie dois objetos da classe *Pessoa*, um representando Albert Einstein (nascido em 14/3/1879) e o outro representando Isaac Newton (nascido em 4/1/1643). Em seguida, mostre quais seriam as idades de Einstein e Newton caso estivessem vivos.

12. Implemente uma classe chamada *Carro* com as seguintes propriedades:

    - Um veículo tem um certo consumo de combustível (medidos em km/litro), uma certa capacidade máxima de combustível e uma certa quantidade de combustível no tanque.
    - O consumo e a capacidade máxima são passados como parâmetro para o 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.
    - Forneça um método *getCombustivel( )*, que retorna o nível atual de combustível.
    - Forneça um método *addCombustivel( )*, para abastecer o tanque.
    - Escreva um pequeno programa que teste sua classe. Exemplo de uso:

```
Carro gol(12, 45); // 12 quilômetros por litro de combustível,
                   //capacidade do tanque é 45 litros
gol.addCombustivel(20); // abastece com 20 litros de combustível.
uno.andar(150); // anda 150 quilômetros.
uno.getCombustivel() // Exibe o combustível que resta no tanque.
```

Fonte:http://www.bernhard.pro.br/disciplinas/java/ensino/java-L01.pdf

13. Implementar a classe PolReg, que define um polígono regular
    - Atributos: número de lados, tamanho do lado
    - Métodos: cálculo do perímetro, cálculo do ângulo interno
    - Construtora que inicializa os valores dos atributos
    - Implementar um programa que receba do usuário o número de lados e o tamanho do lado do polígono e calcule seu ângulo interno e perímetro

14. Implementar a classe TriEq, que define um triângulo equilátero
    - Classe TriEq é derivada de PolReg
    - Atributos: não necessita, estão definidos na base (desde que sejam protegidos)
    - Métodos: cálculo da área + herdados
    - Construtora que inicializa os valores dos atributos (“chama” a construtora da base!)
    - Implementar um programa que receba do usuário o tamanho do lado do triângulo e calcule sua área, perímetro e ângulo interno

15. Adaptar a classe PolReg do exercício 5 para que seja abstrata (método para cálculo da área deve ser abstrato) e implementar a classe Quad, que define um quadrado
    - Classe Quad é derivada de PolReg
    - Atributos: não necessita, estão definidos na base
    - Métodos: cálculo da área + herdados
    - Construtora que inicializa os valores dos atributos (“chama” a construtora da base!)

    Implementar um programa para:

    - Perguntar ao usuário tamanho do lado e número de lados de um polígono qualquer
    - Se for 3 ou 4 lados instanciar um TriEq ou um Quad, respectivamente
    - Calcular e mostrar a área do polígono criado

16. Implementar um sistema de controle de uma conta bancária:
    - Classe Conta: contém os seguintes membros:
        - Atributos: saldo (float) e número (int)
        - Métodos: deposito(), saldo(), retirada(), jurosDiarios() (abstrato)
    - Classe ContaCorrente, derivada de Conta:
        - Implementa jurosDiarios() de 0,1% sobre o que exceder R$ 1.000,00
    - Classe ContaPoupança, derivada de Conta:
        - Implementa jurosDiarios() de 0,2%
    - Aplicação:
        - Menu com opções para: criar conta, ler saldo, depositar, sacar e atualizar saldo em função de dias de aplicação

17. Fazer um sistema de calculadora simples, composto das seguintes classes:

CalcControle: controle da calculadora (“processador”), com os seguintes métodos:

- *public void executar()* – faz a calculadora funcionar através dos seguintes passos:

    - Recebe primeiro operando do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*
    - Recebe segundo operando do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*
    - Recebe operação do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*. Se a operação for igual a 's', finaliza o programa (*System.exit(0)*).
    - Executa a operação (método *opera*) e mostra o resultado através de *CalcInterfac.*
    - Armazena resultado como primeiro operando para a próxima operação e volta para o segundo passo
- *private double opera(double opn1, double opn2, double op)* - executa a operação desejada e retorna o resultado.

CalcDados: implementa a parte da calculadora que armazena os dados (operandos e operação) para o seu funcionamento ("memória"). Possui as seguintes características:

Atributos: dois números do tipo *double* para armazenar os operandos e um dado do tipo *char* para armazenar a operação.

Métodos:

- *public void setOperando(int i, double valor)* – armazena o i-ésimo operando com o valor expresso em *valor*
- *public double getOperando(int i)* – retorna o valor do i-ésimo operando
- *public void setOperacao(char op)* – armazena o caracter *op* como sendo a operação atual
- *public char getOperacao()* – retorna o valor da operação atual

CalcInterface: implementa a parte da calculadora que coleta e exibe informações ao usuário (display e teclado da calculadora). Possui os métodos:

- *public void recebeOperando(int i)* – recebe o operando i da operação (i vale 1 ou 2) e armazena no objeto da classe *CalcDados.*
- *public void recebeOperacao()* – recebe um char representando uma operação válida (+, -, * ou /) e armazena no objeto da classe *CalcDados*
- *public void mostraResultado(double res)* – mostra o resultado recebido como parâmetro.

Criar a classe *Principal*, cujo único objetivo é instanciar os objetos de controle, dados e interface e criar os vínculos (associações) entre eles. Todas as classes citadas devem possuir, além dos atributos citados, outros atributos que representem as referências para os outros objetos (criando as associações entre eles).

18. Escrever a classe *Pessoa* com as seguintes características:
    - atributos: nome e idade
    - métodos: construtora para inicializar os parâmetros e *mostraDados()* que exibe os dados da pessoa no console na forma:

    Nome da pessoa: xxx

Idade da pessoa: yyy

Escrever a classe *Aluno*, derivada de *Pessoa*, com as seguintes características:

- atributos: nome do curso que está cursando
- métodos: construtora para inicializar os atributos e redefinição do método *mostraDados()* para exibir as seguintes mensagens:

Nome do aluno: xxx
Idade do aluno: yyy
Curso do aluno: zzz

Elaborar um programa em Java que:

- declare uma referência para objeto da classe *Pessoa*
- pergunte ao usuário, via console, se ele deseja instanciar um aluno ou uma pessoa
- crie o objeto correspondente, referencie com a referência já criada e chame o método *mostraDados()* para exibir os dados da pessoa ou do aluno

19. Fazer um programa em Java com o seguinte enunciado:
    - Implementar a interface *Classificavel*, que define o método boolean maiorQue(Classificavel outro)
    - Adaptar as classes PolReg, TriEq e Quad para implementar a interface Classificavel
    - Adicionar o atributo "cor" à classe PolReg
    - Implementar uma classe que:
        - Pergunte ao usuário número de lados e tamanho do lado de 5 polígonos
        - Adicione os polígonos a um array
        - Chame o método estático void classifica(Classificavel [] conj) da classe principal para classificar o conjunto por área crescente
        - Mostrar os polígonos classificados (cor, número de lados e área)

20. Escrever a classe Empregado, que pretende representar um empregado em um sistema de RH.
    - Atributos: nome (string), cpf (int) e cargo (string)
    - Construtora que recebe nome, cpf e cargo como parâmetros e preenche os atributos
    - Método *mostraDados()*, que mostra os dados do empregado no formato: *Nome: xxx, CPF: xxx, Cargo: xxx*
    - Método *calculaSalario()*, que recebe o número de horas trabalhadas como parâmetro e retorna 415,00.

    Escrever a classe *Faxineiro*, derivada de *Empregado:*
    - construtora que recebe nome e cpf e repassar os valores pra base
    - método *mostraDados()*, que mostra os dados no formato: *Nome do faxineiro: xxx, CPF: xxx*

- método *calculaSalario()*, que recebe o número de horas trabalhadas e retorna: até 176 horas – 4,50/hora, hora extra – 6,00/hora

Escrever a classe *Gerente*, derivada de *Empregado*

- Atributos: bonus - int.
- construtora que recebe nome, cpf e bonus como parâmetros
- método *mostraDados()*, que mostra os dados no formato: *Nome do gerente: xxx, CPF: xxx, bonus: xxx*
- método *calculaSalario()*, que recebe o número de horas trabalhadas e retorna: até 176 horas – 30,00/hora + bonus, hora extra – não paga

Escrever uma classe de entrada para:

- Perguntar ao usuário se quer inserir os dados de um Faxineiro, Empregado ou Gerente
- Instanciar o objeto correspondente
- Inserir o número de horas trabalhadas e mostrar os dados da pessoa, bem como o salário para aquele mês

Exceções

21. Adaptar o exercício 8 para:
    - lançar a exceção *MenorQueException* caso o número arriscado seja menor do que o sorteado
    - lançar a exceção *MaiorQueException* caso o número arriscado seja maior do que o sorteado
    - capturar essas exceções e tratá-las, mantendo a lógica original.

22. Adaptar o conversor decimal-hexadecimal do exercício 37 para gerar e capturar exceções de:
    - número inválido (tanto em um sentido quanto no outro)
    - valor inválido (não deve aceitar valores negativos)

Fluxos

23. Fazer um programa em Java que:
    - Receba o nome, o código e as duas notas bimestrais de 3 alunos para uma determinada matéria.
    - Salve estes dados em um arquivo. Os dados devem ser salvos registro a registro, obedecendo o seguinte formato:
        i. número inteiro contendo o tamanho em char do nome do aluno.
        ii. sequência de chars correspondente à string que contém o nome do aluno.
        iii. código na forma de número inteiro

iv. duas notas na forma de números inteiros.

Fazer um programa em Java para:

- ler os dados contidos no arquivo gerado pelo programa anterior
- calcular e mostrar: quais alunos foram aprovados, quais foram para exame, quais foram reprovados e a média da turma.

Dica: utilizar *FileOutputStream* encapsulado por um *DataOutputStream.* Utilizar métodos de *String* para convertê-la para um array de bytes.

Coleções

24. Fazer um programa para:

- Criar a classe *Item*, que pretende representar um item sendo comprado em um supermercado.
  - Atributos: nome do item, valor unitário e quantidade
  - Métodos:
    - construtora que recebe como parâmetro e ajusta os valores dos atributos
    - *getNome()*
    - *getValorTotal()*
- Criar a classe *ListaDeCompras*, que permite criar uma lista de compras em um supermercado.
  - Atributos: ArrayList de *Item.*.
  - Métodos:
    - *adicionarItem(Item i)*
    - *removerItem(Item i)*
    - *getValorTotal()*
- Criar a classe *Principal*, com o objetivo de criar itens, uma lista de compras e exercitar os métodos.

25. Fazer uma classe *Aluno* que possua as seguintes características:

- dois atributos do tipo inteiro: primeira nota parcial (de 0 a 100) e Segunda nota parcial (de 0 a 100)
- um atributo *String* representando o nome do aluno
- possua métodos para ler e escrever os atributos (ou uma construtora)

Fazer uma classe *Controle* que:

- pergunte ao usuário o nome e as duas notas parciais de um aluno. Caso o nome entrado seja “fim” isso significa que o usuário não quer inserir mais nenhum aluno, do contrário deve ser instanciado um objeto da classe Aluno e armazenados os dados digitados.

Dicas: usar um objeto da classe *ArrayList* de Java para armazenar as referências para os objetos instanciados). Usar o método *equals* da classe *String* para verificar se o valor do nome entrado é igual a "fim".

- Calcular, ao final da inserção de todos os alunos, a média da turma, quantos alunos foram aprovados, quantos foram para a final e quantos foram reprovados e mostrar os códigos de todos os alunos cujas notas ficaram abaixo da média da turma.

26. Fazer uma classe *ListNode* que represente os nós de uma lista encadeada. Esta classe deve conter o campo nome e uma referência para o próximo nó da lista chamada prox.
    Fazer então a classe *ListOperation* que implementa os seguintes métodos:
    - *add(ListNode n, ListNode ant)* – adiciona o nó n após o nó ant (no início se ant for 0).
    - *remove(ListNode n)* – remove o nó n da lista.
    - *print()* – percorre a lista e mostra o valor do campo nome de cada nó.

    Fazer uma classe principal que exercita as operações acima implementadas. Tentar efetuar as mesmas operações utilizando LinkedList e comparar o desempenho.

27. Implementar um "teste de memorização" utilizando coleções
    - Coleção é um *ArrayList* ou *LinkedList* com 10 nomes de algo colecionável (p. ex., 10 modelos de automóvel)
    - Ao iniciar o programa, 5 elementos da coleção são mostrados ao usuário em um frame, em uma ordem qualquer, durante 4 segundos
        - Dica: usar o método *Collections.shuffle()*
    - Os elementos são ocultados e o usuário deve entrar os elementos em uma caixa de texto, na ordem em que foram exibidos e separados por vírgulas
    - programa informa ao usuário se acertou ou não

28. Fazer um programa para
    - Obter o conteúdo da tabela "TabAlunos" da base "teste_db"
        - Cada registro deve ser armazenado em um objeto da classe Aluno
    - Inserir cada objeto em dois TreeSet:
        - Primeiro: comparação por nome (ordem alfabética)
        - Segundo: comparação por coeficiente
    - Exibir os conteúdos de cada TreeSet

API Gráfica

29. Implementar o layout de uma calculadora
    - Caixa de texto com o número no alto do frame
    - Números de 0 a 9 e sinais de . e = em grid de 4x3 no centro do frame
    - Botões de + e – à direita do frame

- Botões de * e / na parte de baixo do frame

30. Implementar um conversor Celsius- Fahrenheit com as seguintes características:
    - Interface: duas caixas de texto para receber valores de temperatura em Celsius e Fahrenheit e botões para a conversão em ambos os sentidos
    - Funcionalidade: clicando-se um botão deve-se fazer o cálculo no sentido correspondente. Considerar que o usuário SEMPRE insere números e que caixa vazia significa 0.
    - C = (F - 32)*5/9
    - F = C*9/5 + 32

31. Implementar um cadastro/status de RH
    - Interface:
        - Caixa de texto para o nome do funcionário
        - Dois botões de rádio para escolher entre “Gerente” e “Não gerente”
        - Caixa de texto para inserção das horas trabalhadas
        - Menu “Ações” com opções “Inserir” e “Visualizar”
    - Funcionalidade
        - O usuário pode inserir novos valores e clicar no item “Inserir” do menu. Após o décimo valor inserido deve mostrar uma mensagem (diálogo) de erro
        - O usuário pode abrir uma nova janela para exibir os valores de maior e menor salário e nome dos respectivos empregados clicando no item “Visualizar” do menu

32. Adaptar o programa de cadastro/status de RH anterior para:
    - Incluir o método “validaNome”, que lança uma exceção de nome inválido se este for nulo ou se for formado por menos de duas palavras
    - Lançar uma exceção de salário inválido se este não for um valor numérico inteiro
    - Capturar as exceções e mostrar um diálogo de erro

33. Adaptar o programa de cadastro/status de RH anterior para:
    - Incluir as opções “Abrir” e “Salvar” no menu “Ações”
    - Opção “Salvar” serializa os objetos das classes Empregado/Gerente em um arquivo
    - Opção “Abrir” recupera os cadastros serializados

34. Implementar um programa em Java para:
    - Oferecer um menu de opções: inserir compra, carregar, salvar e mostrar
    - Ao clicar em “inserir compra”, o usuário deve fornecer um nome de item (máx. 20 caracteres) e um double representando o preço
    - Ao clicar em “salvar”, o programa deve salvar os nomes de itens e preços em um arquivo utilizando fluxo de saída (dica: utilizar DataOutputStream)

- Ao clicar em “carregar”, o programa deve abrir o arquivo com as últimas compras e carregá-lo em memória (dica: utilizar DataInputStream)
- Ao clicar em “mostrar”, o programa deve exibir os nomes de itens e preços atualmente em memória

35. Implementar um aplicativo de gerenciamento de login com as seguintes funcionalidades:

- item de menu “Cadastrar login”, o qual exibe um diálogo que permite cadastrar um novo usuário com uma nova senha. Dica: usar JpasswordField para receber a senha.
- item de menu “Efetuar login”, que exibe um diálogo no qual o usuário deve fornecer um nome e uma senha. Caso seu nome e senha estejam cadastrados deve ser exibido um diálogo de “Acesso autorizado”, do contrário deve ser exibido um diálogo de “Acesso negado”.

Para cada novo login cadastrado o programa deve atualizar um arquivo de informações de login e senha. Ao ser iniciado o aplicativo deve verificar se esse arquivo existe e, em caso afirmativo, carregar os nomes e senhas já cadastrados para poderem ser usados na verificação de login.

36. Implementar um jogo de “Clique rápido”
    - Janela possui 3 botões e dois label:
        - Label 1: mostra o botão a ser clicado
        - Label 2: mostra o tempo acumulado
    - Quando label1 mostra um botão (p. ex., “Botão 1”), o usuário deve clicar no botão correspondente o mais rápido possível
        - Número do botão e atraso é obtido aleatoriamente
    - Tempo acumulado é mostrado no label2
    - Jogo finaliza após cinco cliques

37. Implementar um conversor hexadecimal - decimal com as seguintes características:
    - Interface: duas caixas de texto para receber valores numéricos na base hexadecimal e na base decimal e botões para a conversão em ambos os sentidos
    - Funcionalidade: clicando-se um botão deve-se fazer o cálculo no sentido correspondente. Considerar que caixa vazia ou valor inválido deve ser convertido para 0.

38. Implementar um gerador de lista de músicas utilizando Botão (ou menu) para abrir um diálogo de escolha de diretório
    - Aplicativo deve criar um arquivo texto (lista.txt) contendo o conteúdo do diretório, um arquivo por linha.
    - Cada subdiretório encontrado deve ter o seu conteúdo listado, acrescentando uma tabulação no início da linha.
    - O programa deve listar somente arquivos .MP3 e .WAV

Threads

39. Faça um programa em Java com as seguintes características:
    - o programa pede um palpite ao usuário: “Cara” ou “Coroa”
    - o programa possui 2 threads, sendo que uma delas imprime a palavra “Cara” e a outra imprime a palavra “Coroa” no console
    - cada uma das threads “dorme” por um tempo aleatório entre 0 e 30 ms
    - ao pressionar-se enter, o programa principal deve finalizar as threads e verificar se a última palavra que foi mostrada corresponde ao palpite. Se corresponde mostra “Acertou”, do contrário mostra “Errou”.

40. Pretende-se simular a dinâmica de uma corrida de Fórmula 1 através de duas threads, uma delas representando o carro de Felipe Massa e a outra o carro de Lewis Hamilton. Para tanto as threads têm as seguintes características:
    - cada thread possui um loop com 65 iterações, simulando uma corrida de 65 voltas
    - a cada volta a thread dorme durante um tempo aleatório entre 0 e 1 s. Este tempo representa o tempo gasto para percorrer a volta.

O programa deve aguardar as duas threads terminarem (dica: usar *join()*) , representando que os dois carros cruzaram a linha de chegada, e então mostrar uma mensagem informando qual dos 2 pilotos venceu a corrida.

41. Implementar um programa para simular a ocorrência de transações bancárias
    - Banco possui 10 contas correntes e um total de R$ 100.000,00
    - Banco possui 5 correntistas (threads) que movimentam recursos de uma conta para outra aleatoriamente
    - Janela do programa exibe:
        - Saldo de cada conta
        - Saldo total do banco
    - O programa deve tomar providências para que o saldo total do banco seja sempre de R$ 100.000,00, independente de quais transações ocorreram.

42. Implementar um simulador de prédio com elevadores.
    - Prédio possui 6 andares e 2 elevadores
    - Capacidade máxima de cada elevador é 4 pessoas
    - Prédio é implementado na forma de um frame, com um botão de “Subir” e outro de “Descer” em cada andar.

- A cada clique do botão é incrementado um contador, significando quantas pessoas naquele andar gostariam de subir/descer.
- Cada elevador deve ser movimentado por uma thread, que verifica na central de elevadores quais as requisições pendentes e toma ou não a iniciativa de atendê-las.
- Quando um elevador pára em um andar para atender requisições, o programa deve parar a simulação e permitir que as pessoas embarcando no elevador selecionem seus andares de destino.

43. Deseja-se implementar um simulador do cruzamento de duas ruas de mão única, nas quais trafega um número aleatório de veículos. Para tanto, fazer um programa em Java com as seguintes características:

- uma thread responsável por controlar o funcionamento dos semáforos. Esta thread sorteia um valor aleatório entre 10 e 15 s para o tempo em verde de cada um dos semáforos do cruzamento, sendo o tempo em amarelo é sempre de 2 s. A cada mudança de estado de um dos semáforos esta thread deve imprimir uma mensagem no console.
- uma thread para cada rua, responsável por simular o tráfego naquela rua. Cada uma destas threads tem as seguintes características:
    a. adiciona carros a um *ArrayList* em intervalos de tempo aleatórios entre 1 e 2 s, quando o semáforo da sua rua está fechado, e imprime uma mensagem no console informando que um carro foi adicionado e o número da placa.
        i. cada carro é um objeto da classe *Carro* que possui um número de placa aleatório entre 0 e 9999, o qual é armazenado em um atributo pela construtora.
    b. remove os carros do mesmo *ArrayList*, também em intervalos de tempo aleatórios entre 1 e 2 s, quando o semáforo está aberto, e imprime uma mensagem no console informando que um carro passou pelo cruzamento e o número da placa.
- uma thread responsável por informar o status do tráfego. Esta thread informa, a cada 5 s, quantos carros estão parados em cada rua do cruzamento naquele momento.
- uma thread que aguarda o usuário digitar uma tecla e termina o programa. Esta thread precisa aguardar todas as outras terminarem antes de ela própria terminar.

JDBC

44. Implementar um aplicativo para
    - mostrar o conteúdo da tabela “TabAlunos” da base de dados “teste_db” em um JTable ou na forma de uma lista no console

- O aplicativo deve conter botões (ou opções de menu no console) para classificar os registros visualizados por nome e por curso
- O aplicativo deve permitir inserir um novo registro na tabela

Sockets

45. Escrever dois programas em Java com as seguintes características:

- o programa 1 cria um socket UDP na porta 2000 e aguarda o recebimento de pacotes
- o programa 2 envia um pacote com uma string para o programa 1, através de socket UDP, e aguarda uma resposta.
- o programa 1 inverte o conteúdo do pacote recebido e devolve este conteúdo para o programa 2, que recebe e mostra no console

46. Implementar um cliente de chat por UDP
    - Cliente se conecta ao servidor na porta 125
    - Cliente possui uma interface gráfica:
        - Campo de texto superior ⇒ mensagens recebidas
        - Campo de texto inferior ⇒ mensagens a serem enviadas
        - Lista com os clientes cadastrados no momento
        - Botão “Enviar”
    - Protocolo com o servidor:
        - CONNECT *nome* ⇒ conecta-se usando um nome (recebe OK como resposta)
        - MSG(*dest*) *msg* ⇒ envia a mensagem *msg* para o destinatário *dest* (recebe OK ou FAIL como resposta)
        - GET ⇒ recebe como resposta uma lista dos clientes conectados, linha por linha (primeira linha contém o texto CLIENTS)
        - MSGFROM(*rem*) *msg* ⇒ mensagem recebida de outro cliente
        - ALIVE ⇒ indica que o cliente ainda está vivo

47. Deseja-se implementar um sistema eletrônico de votação a partir de uma aplicação servidora e de aplicações clientes que se conectam a ela através de sockets de rede. O sistema tem as seguintes características:
    - A aplicação servidora, ao ser iniciada, solicita que o usuário entre com o número de candidatos e os códigos dos candidatos via console.
    - A aplicação servidora passa a aceitar conexões TCP na porta 909 e pacotes UDP na porta 939.
    - Quando uma aplicação cliente é iniciada, esta abre uma conexão com o servidor, via porta TCP 909, recebe do servidor uma lista com os códigos dos candidatos (array de *int*) e fecha a conexão
    - A aplicação cliente passa a receber do usuário, via console de texto, o código do candidato a ser votado. Quando isto acontece, a aplicação cliente envia para o

servidor, via porta UDP 939, um pacote contendo um número *int* representando o código do candidato que foi votado.
- A aplicação servidora, ao receber um novo pacote UDP com um voto, atualiza o resultado da votação e mostra no console.

Miscelâneas

48. Empacotar em JAR e testar alguma das aplicações já desenvolvidas em sala de aula (exemplos ou exercícios). Utilizar linha de comando e também wizard da ferramenta de desenvolvimento.

49. Implementar uma aplicação que:
    - Possui uma janela que pode conter quadros internos
    - Possui dois quadros internos:
        - Um quadro contendo uma área de texto, na qual se pode escrever
        - Um quadro contendo informações de status do texto: número de caracteres, número de palavras e número de vogais

50. Adaptar o jogo do “Clique rápido” para funcionar como uma applet.

51. Implementar um “relógio universal”. Relógio possui:
    - Texto estático para mostrar a data e hora atuais (incluindo dia da semana)
    - Lista contendo todas as possibilidades de localização de data e hora
        - Localizações instaladas são obtidas a partir de getAvailableLocales()