Conteúdo

Lista 1: Java, classes e polimorfismo

Exercício 1.1: Comandos do mouse

Escreva um programa java que mostre ao usuário uma janela que contém um painel capaz de receber comandos com o mouse para definir o ponto central de um circulo e seu raio. Enquanto o usuário arrasta o mouse para desenhar o circulo, em uma barra na base da janela devem ser mostrados os valores do ponto central do circulo, o valor de seu raio, o valor de sua área e o valor de sua circunferência.

Exercício 1.2: Armazenar comandos do mouse

Escreva uma classe a ser acrescentada ao programa do exercício 1.1, para gravar os valores correspondentes ao ponto central e ao raio de uma série de círculos definidos pelo usuário em uma janela que contém um painel capaz de receber comandos com o mouse para definir o ponto central de um circulo e seu raio. Em quanto o usuário arrasta o mouse para desenhar o circulo, em uma barra na base da janela devem ser mostrados os valores do ponto central do circulo, o valor de seu raio, o valor de sua área e o valor de sua circunferência.

Exercício 1.3: Desenhos

Escreva um programa java que mostre ao usuário uma janela contendo um painel de desenho, uma barra de mensagens na parte inferior da janela e uma barra de botões na parte superior da janela. Nesta barra, devem ser apresentadas opções para selecionar os modos de desenho correspondentes a pontos, círculos e retângulos. Cada tipo de desenho deve ser armazenado em um vetor com pelo menos 500 posições. Na barra de mensagens, deve ser mostrado um texto informando as características de cada desenho conforme o usuário os realiza com o auxílio do mouse.

Exercício 1.4: Bits Trocados

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

As Ilhas Weblands formam um reino independente nos mares do Pacífico. Como é um reino recente, a sociedade é muito influenciada pela informática. A moeda oficial é o Bit; existem notas de B$ 50,00, B$10,00, B$5,00 e B$1,00. Você foi contratado(a) para ajudar na programação dos caixas automáticos de um grande banco das Ilhas Weblands.

Tarefa

Os caixas eletrônicos das Ilhas Weblands operam com todos os tipos de notas disponíveis, mantendo um estoque de cédulas para cada valor (B$ 50,00, B$10,00, B$5,00 e B$1,00). Os clientes do banco utilizam os caixas eletrônicos para efetuar retiradas de um certo número inteiro de Bits.

Sua tarefa é escrever um programa que, dado o valor de Bits desejado pelo cliente, determine o número de cada uma das notas necessário para totalizar esse valor, de modo a minimizar a quantidade de cédulas entregues. Por exemplo, se o cliente deseja retirar B$50,00, basta entregar uma única nota de cinquenta Bits. Se o cliente deseja retirar B$72,00, é necessário entregar uma nota de B$50,00, duas de B$10,00 e duas de B$1,00.

Entrada

A entrada é um valor inteiro positivo V, que indica o valor solicitado pelo cliente. O final da entrada é indicado por V = 0.

Exemplo de Entrada

```
1
72
0
```

Saída

Para cada valor da entrada seu programa deve produzir uma linha na saída, na qual devem aparecer quatro inteiros I, J, K e L que representam o resultado encontrado pelo seu programa: I indica o número de cédulas de B$50,00, J indica o número de cédulas de B$10,00, K indica o número de cédulas de B$5,00 e L indica o número de cédulas de B$1,00.

Exemplo de Saída

```
0 0 0 1

1 2 0 2
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq V \leq 10000$ ($V = 0$ apenas para indicar o fim da entrada)

Observações

O seu programa deve possuir uma interface gráfica com o usuário, na qual um valor de saque é solicitado ao usuário e em seguida o número de notas de cada valor deve ser apresentado. Ao ser fornecido o valor 0 o programa deve ser encerrado.

Exercício 1.5: Macaco-prego

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

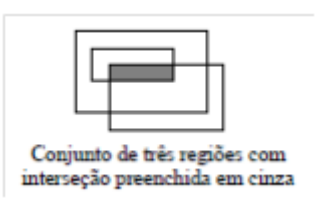
O macaco-prego é um animal irrequieto e barulhento, merecedor também dos adjetivos desordeiro e despudorado. A sua cabeça, encimada por uma densa pelagem negra ou marrom-escura, semelhante a um gorro, torna seu aspecto inconfundível. Apesar de ser o macaco mais comum nas matas do país, uma de suas sub-espécies encontra-se seriamente ameaçada de extinção: o macaco-prego-do-peito-amarelo, que se distingue das demais pela coloração amarelada do peito e da parte anterior dos braços.

Um grande esforço foi feito pelos primatologistas para aumentar a população dos macacos-prego-do-peito-amarelo. Sabe-se que eles se alimentam de plantas, das quais

consomem preferencialmente frutos e brotos. Alimentam-se também de muitos animais, preferencialmente lesmas, lagartas e rãs, e preferem as florestas mais densas. Para determinar o melhor local do país para criar uma nova reserva ambiental para os macacos-prego-do-peito-amarelo, o governo fez um levantamento das regiões no país onde as condições preferidas desses animais ocorrem: regiões de floresta densa, regiões com frutos, regiões com muitos brotos, etc. Ajude a salvar os macacos-prego-do-peito-amarelo.

Tarefa

As regiões propícias para o macaco-prego-do-peito-amarelo foram determinadas como retângulos cujos lados são todos verticais ou horizontais. Sua tarefa é encontrar o local ideal para a reserva ambiental, definida como a interseção de todas as regiões dadas.



Conjunto de três regiões com interseção preenchida em cinza

Conjunto de três regiões com interseção vazia

As regiões foram divididas de tal forma que uma região não tangencia qualquer outra região. Assim, a interseção entre quaisquer duas regiões ou é um retângulo ou é vazia.

Entrada

Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, N, que indica o número de regiões (o valor $N = 0$ indica o final da entrada). Seguem-se N linhas, cada uma contendo quatro números inteiros X, Y, U e V que descrevem uma região: o par X, Y representa a coordenada do canto superior esquerdo e o par U, V representa a coordenada do canto inferior direito de um retângulo.

Exemplo de Entrada

```
3
0 6 8 1
1 5 6 3
2 4 9 0
3
0 4 4 0
3 1 7 -3
6 4 10 0
0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter as coordenadas do retângulo de interseção encontrado pelo seu programa, no mesmo formato utilizado na entrada. Caso a interseção seja vazia, a segunda linha deve conter a expressão "nenhum".

A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
2 4 6 3
Teste 2
nenhum
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)
$-10000 \leq X \leq 10000$
$-10000 \leq Y \leq 10000$
$-10000 \leq U \leq 10000$
$-10000 \leq V \leq 10000$

Exercício 1.6: Quermesse

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

Os alunos do último ano resolveram organizar uma quermesse para arrecadar fundos para a festa de formatura. A festa prometia ser um sucesso, pois o pai de um dos formandos, Teófilo, dono de uma loja de informática, decidiu doar um computador para ser sorteado entre os que comparecessem. Os alunos prepararam barracas de quentão, pipoca, doces, ensaiaram a quadrilha e colocaram à venda ingressos numerados sequencialmente a partir de 1. O número do ingresso serviria para o sorteio do computador. Ficou acertado que Teófilo decidiria o método de sorteio; em princípio o sorteio seria, claro, computadorizado.

O local escolhido para a festa foi o ginásio da escola. A entrada dos participantes foi pela porta principal, que possui uma roleta, onde passa uma pessoa por vez. Na entrada, um funcionário inseriu, em uma lista no computador da escola, o número do ingresso, na ordem de chegada dos participantes. Depois da entrada de todos os participantes, Teófilo começou a trabalhar no computador para preparar o sorteio. Verificando a lista de presentes, notou uma característica notável: havia apenas um caso, em toda a lista, em que o participante que possuía o ingresso numerado com i, havia sido a i-ésima pessoa a entrar no ginásio. Teófilo ficou tão encantado com a coincidência que decidiu que o sorteio não seria necessário: esta pessoa seria o ganhador do computador.

Tarefa

Conhecendo a lista de participantes, por ordem de chegada, sua tarefa é determinar o número do ingresso premiado, sabendo que o ganhador é o único participante que tem o número do ingresso igual à sua posição de entrada na festa.

Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro positivo N que indica o número de participantes da

festa. A linha seguinte contém a sequência, em ordem de entrada, dos N ingressos das pessoas que participaram da festa. O final da entrada é indicado quando N = 0. Para cada conjunto de teste da entrada haverá um único ganhador.

Exemplo de Entrada

```
4
4 5 3 1
10
9 8 7 6 1 4 3 2 12 10
0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas. A primeira linha identifica o conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter o número do ingresso do ganhador, conforme determinado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
3
Teste 2
10
```

(esta saída corresponde ao exemplo de entrada acima) **Restrições**

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)

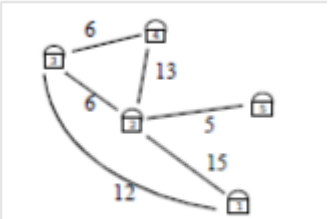
Exercício 1.7: Rede ótica

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

Os caciques da região de Tutuaçu pretendem integrar suas tribos à chamada "aldeia global". A primeira providência foi a distribuição de telefones celulares a todos os pajés. Agora, planejam montar uma rede de fibra ótica interligando todas as tabas. Esta empreitada requer que sejam abertas novas picadas na mata, passando por reservas de flora e fauna. Conscientes da necessidade de preservar o máximo possível o meio ambiente, os caciques encomendaram um estudo do impacto ambiental do projeto. Será que você consegue ajudá-los a projetar a rede de fibra ótica?

Tarefa

Vamos denominar uma ligação de fibra ótica entre duas tabas de um ramo de rede. Para possibilitar a comunicação entre todas as tabas é necessário que todas elas estejam interligadas, direta (utilizando um ramo de rede) ou indiretamente (utilizando mais de um ramo). Os caciques conseguiram a informação do impacto ambiental que causará a construção dos ramos. Alguns ramos, no entanto, nem foram considerados no estudo ambiental, pois sua construção é impossível.



Ramos de rede possíveis com custo ambiental associado

Interligação das tabas com menor custo ambiental

Sua tarefa é escrever um programa para determinar quais ramos devem ser construídos, de forma a possibilitar a comunicação entre todas as tabas, causando o menor impacto ambiental possível.

Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos N e M que indicam, respectivamente, o número de tabas e o número de ramos de redes possíveis. As tabas são numeradas de 1 a N. As M linhas seguintes contêm três inteiros positivos X, Y e Z, que indicam que o ramo de rede que liga a taba X à taba Y tem impacto ambiental Z. Com os conjuntos de teste dados sempre é possível interligar todas as tabas. O final da entrada é indicado quando $N = 0$.

Exemplo de Entrada

```
3 3
1 2 10
2 3 10
3 1 10
5 6
1 2 15
1 3 12
2 4 13
2 5 5
3 2 6
3 4 6
0 0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir uma lista dos ramos de redes que devem ser construídos. A lista deve ser precedida de uma linha que identifica o conjunto de teste, no formato ”Teste n”, onde n é numerado a partir de 1. A lista é composta por uma sequência de ramos a serem construídos, um ramo por linha. Um ramo é descrito por um par de tabas X e Y, com X ¡ Y. Os ramos de rede podem ser listados em qualquer ordem, mas não deve haver repetição. Se houver mais de uma solução possível, imprima apenas uma delas. O final de uma lista de ramos deve ser

marcado com uma linha em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
1 2
1 3
Teste 2
1 3
2 3
2 5
3 4
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq N \leq 100$ ($N = 0$ apenas para indicar o fim da entrada)
$1 \leq M \leq N(N-1)/2$
$1 \leq X \leq 100$
$1 \leq Y \leq 100$
$1 \leq Z \leq 100$

Exercício 1.8: Saldo de gols

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

Hipólito é um torcedor fanático. Coleciona flâmulas, bandeiras, recortes de jornal, figurinhas de jogadores, camisetas e tudo o mais que se refira a seu time preferido. Quando ganhou um computador de presente em uma festa, resolveu montar um banco de dados com os resultados de todos os jogos de seu time ocorridos desde a sua fundação, em 1911. Depois de inseridos os dados, Hipólito começou a ficar curioso sobre estatísticas de desempenho do time. Por exemplo, ele deseja saber qual foi o período em que o seu time acumulou o maior saldo de gols. Como Hipólito tem o computador há muito pouco tempo, não sabe programar muito bem, e precisa de sua ajuda.

Tarefa

É dada uma lista, numerada sequencialmente a partir de 1, com os resultados de todos os jogos do time (primeira partida: 3 x 0, segunda partida: 1 x 2, terceira partida: 0 x 5 ...). Sua tarefa é escrever um programa que determine em qual período o time conseguiu acumular o maior saldo de gols. Um período é definido pelos números de sequência de duas partidas, A e B, onde $A \leq B$. O saldo de gols acumulado entre A e B é dado pela soma dos gols marcados pelo time em todas as partidas realizadas entre A e B (incluindo as mesmas) menos a soma dos gols marcados pelos times adversários no período. Se houver mais de um período com o mesmo saldo de gols, escolha o maior período (ou seja, o período em que B - A é maior). Se ainda assim houver mais de uma solução possível, escolha qualquer uma delas como resposta.

Entrada

Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, N, que indica o número de partidas realizadas pelo time (o valor N = 0 indica o final da entrada). Seguem-se N linhas, cada uma contendo um par de números inteiros não negativos X e Y que representam o resultado da partida: X são os gols a favor e Y os gols contra o time de Hipólito. As partidas são numeradas sequencialmente a partir de 1, na ordem em que aparecem na entrada.

Exemplo de Entrada

```
2
2 3
7 1
9
2 2
0 5
6 2
1 4
0 0
5 1
1 5
6 2
0 5
3
0 2
0 3
0 4
0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter um par de inteiros I e J que indicam respectivamente a primeira e última partidas do melhor período, conforme determinado pelo seu programa, exceto quando o saldo de gols do melhor período for menor ou igual a zero; neste caso a segunda linha deve conter a expressão "nenhum". A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
2 2
Teste 2
3 8
Teste 3
nenhum
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)
$1 \leq A \leq N$
$A \leq B \leq N$
$0 \leq X \leq 50$
$0 \leq Y \leq 50$

Lista 2: Arquivos

Exercício 2.1: Desenhos

Escreva um programa java que mostre ao usuário uma janela (GUI) contendo um painel de desenho, uma barra de mensagens na parte inferior da janela e uma barra de botões na parte superior da janela. Nesta barra, devem ser apresentadas opções para selecionar os modos de desenho correspondentes a pontos, círculos e retângulos, um botão para gravar e outro para ler as informações de um arquivo utilizado para armazenar os dados do desenho. Cada tipo de desenho deve ser armazenado em um vetor com pelo menos 500 posições. Na barra de mensagens, deve ser mostrado um texto informando as características de cada desenho conforme o usuário os realiza com o auxílio do mouse. Ao clicar no botão para gravar, as informações de todos os desenhos feitos pelo usuário devem ser gravadas no arquivo, e ao clicar no botão para ler, as informações dos desenhos devem ser descartadas e substituídas pelas informações contidas no arquivo.

Exercício 2.2: Lista de arquivos

Escreva um programa java que solicite ao usuário o caminho para uma pasta em um dispositivo de armazenamento em massa e, a partir desta pasta, leia todos os nomes de arquivos e pastas aí contidas. Para cada pasta encontrada o processo deve ser repetido. Na interface gráfica com o usuário (GUI) do seu programa deve ser apresentado ao usuário o total de arquivos e de pastas encontradas e uma opção para que a estrutura encontrada seja mostrada de forma hierárquica.

Exercício 2.3: Notas de alunos

1. Elabore um programa que armazene em um arquivo o nome e a nota de n alunos.
2. Leia o arquivo com os nomes e as notas dos alunos e mostre:
    - o nome do aluno que obteve a maior nota;
    - o nome do aluno com a menor nota;
    - a média das notas da sala.

Exercício 2.4: Matriz em formato texto

Faça um programa que seja capaz de ler os valores de uma matriz $n \times m$ do usuário e salve-a em um arquivo de texto. Depois, leia o arquivo e guarde os valores em uma nova matriz e mostre a matriz recuperada.

Exercício 2.5: Matriz em formato binário

Faça um programa que seja capaz de ler os valores de uma matriz $n \times m$ do usuário e salve-a em um arquivo binário. Depois, leia o arquivo e guarde os valores em uma nova matriz e mostre a matriz recuperada.

Exercício 2.6: Criptografia simples

Elabore um programa simples para criptografar um arquivo de texto. O programa deverá receber do usuário o nome do arquivo a ser criptografado e um valor inteiro de deslocamento. De forma que cada caractere será deslocado de acordo com o fator de deslocamento.

Exemplos:

Arquivo de entrada: "1 2 3 4"; Fator de deslocamento = 1; Saída: "2 3 4 5"

Arquivo de entrada: "1 2 3 4"; Fator de deslocamento = 3; Saída: "4 5 6 7"

Arquivo de entrada: "1 2 3 4"; Fator de deslocamento = 10; Saída: "11 12 13 14"

Salve o texto criptografado em um novo arquivo binário. Depois faça a operação inversa, leia o arquivo criptografo e utilize o fator de deslocamento para decifrar o texto.

Exercício 2.7: Corretor de provas

Faça um verificador de respostas para auxiliar na correção de provas. Para isto, utilize um arquivo como gabarito, onde cada linha representa a resposta correta para uma questão; e outro arquivo, correspondente às respostas dadas por um aluno. Considere que existem 5 opções para cada questão da prova: A, B, C, D, E. Imprima a quantidade de acertos do aluno.

Exercício 2.8: Bits Trocados

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

As Ilhas Weblands formam um reino independente nos mares do Pacífico. Como é um reino recente, a sociedade é muito influenciada pela informática. A moeda oficial é o Bit; existem notas de B$ 50,00, B$10,00, B$5,00 e B$1,00. Você foi contratado(a) para ajudar na programação dos caixas automáticos de um grande banco das Ilhas Weblands.

Tarefa

Os caixas eletrônicos das Ilhas Weblands operam com todos os tipos de notas disponíveis, mantendo um estoque de cédulas para cada valor (B$ 50,00, B$10,00, B$5,00 e B$1,00). Os clientes do banco utilizam os caixas eletrônicos para efetuar retiradas de um certo número inteiro de Bits.

Sua tarefa é escrever um programa que, dado o valor de Bits desejado pelo cliente, determine o número de cada uma das notas necessário para totalizar esse valor, de modo a minimizar a quantidade de cédulas entregues. Por exemplo, se o cliente deseja retirar B$50,00, basta entregar uma única nota de cinquenta Bits. Se o cliente deseja retirar B$72,00, é necessário entregar uma nota de B$50,00, duas de B$10,00 e duas de B$1,00.

Entrada

A entrada é composta de vários conjuntos de teste. Cada conjunto de teste é composto por uma única linha, que contém um número inteiro positivo V, que indica o valor solicitado pelo cliente. O final da entrada é indicado por V = 0. Exemplo de Entrada

```
1
72
0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. Na segunda linha devem aparecer quatro inteiros I, J, K e L que representam o resultado encontrado pelo seu programa: I indica o número de cédulas de B$50,00, J indica o número de cédulas de B$10,00, K indica o número de cédulas de B$5,00 e L indica o número de cédulas de B$1,00. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
0 0 0 1
Teste 2
1 2 0 2
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq V \leq 10000$ ($V = 0$ apenas para indicar o fim da entrada)

Exercício 2.9: Meteoros

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

Em noites sem nuvens pode-se muitas vezes observar pontos brilhantes no céu que se deslocam com grande velocidade, e em poucos segundos desaparecem de vista: são as chamadas estrelas cadentes, ou meteoros. Meteoros são na verdade partículas de poeira de pequenas dimensões que, ao penetrar na atmosfera terrestre, queimam-se rapidamente (normalmente a uma altura entre 60 e 120 quilômetros). Se os meteoros são suficientemente grandes, podem não queimar-se completamente na atmosfera e dessa forma atingem a superfície terrestre: nesse caso são chamados de meteoritos.

Zé Felício é um fazendeiro que adora astronomia e descobriu um portal na Internet que fornece uma lista das posições onde caíram meteoritos. Com base nessa lista, e conhecendo a localização de sua fazenda, Zé Felício deseja saber quantos meteoritos caíram dentro de sua propriedade. Ele precisa de sua ajuda para escrever um programa de computador que faça essa verificação automaticamente.

Tarefa
São dados:

- uma lista de pontos no plano cartesiano, onde cada ponto corresponde à posição onde caiu um meteorito;
- as coordenadas de um retângulo que delimita uma fazenda.

As linhas que delimitam a fazenda são paralelas aos eixos cartesianos. Sua tarefa é escrever um programa que determine quantos meteoritos caíram dentro da fazenda (incluindo meteoritos que caíram exatamente sobre as linhas que delimitam a fazenda).

Entrada

Seu programa deve ler vários conjuntos de testes. A primeira linha de um conjunto de testes quatro números inteiros X_1, Y_1, X_2 e Y_2, onde (X_1, Y_1) é a coordenada do canto superior esquerdo e (X_2, Y_2) é a coordenada do canto inferior direito do retângulo que delimita a fazenda. A segunda linha contém um inteiro, N, que indica o número de meteoritos. Seguem-se N linhas, cada uma contendo dois números inteiros X e Y, correspondendo às coordenadas de cada meteorito. O final da entrada é indicado por $X_1 = Y_1 = X_2 = Y_2 = 0$.

Exemplo de Entrada

```
2 4 5 1
2
1 2
3 3
2 4 3 2
3
1 1
2 2
3 3
0 0 0 0
```

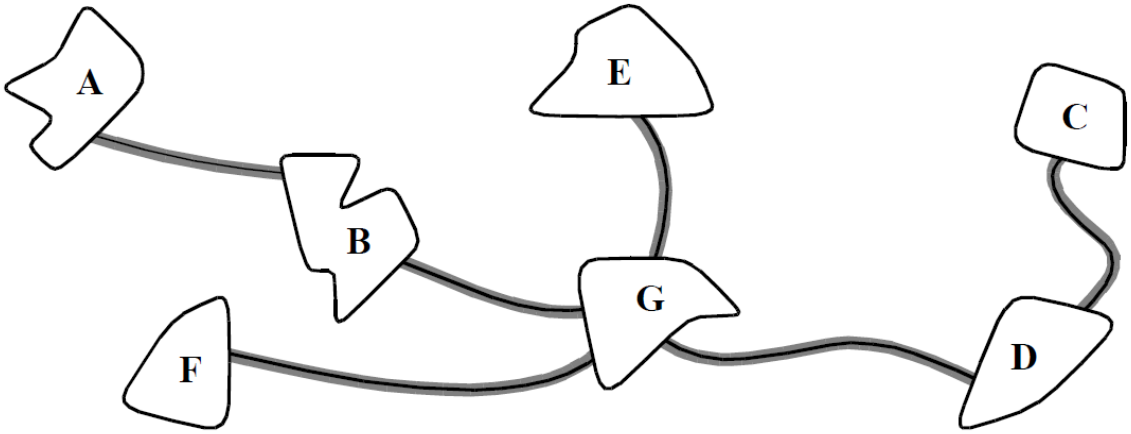
Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter o número de meteoritos que caíram dentro da fazenda. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
1
Teste 2
2
```

Figura 1: Sete vilas interligadas por linhas de trem



(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq N \leq 10.000$
$0 \leq X \leq 10.000$
$0 \leq Y \leq 10.000$
$0 \leq X1 < X2 \leq 10.000$
$0 \leq Y2 < Y1 \leq 10.000$

Exercício 2.10: Dengue

(Exercício retirado/adaptado da Olimpíada Brasileira de Informática.)

A Costa do Mosquito é um país pequeno mas paradisíaco. Todos os habitantes têm boas moradias, bons empregos, o clima é agradável, e os governantes são justos e incorruptíveis. O sistema de transporte público da Costa do Mosquito é composto de uma rede de linhas de trem. O sistema foi projetado de forma peculiar: existe um único percurso ligando duas quaisquer vilas (esse percurso possivelmente passa por outras vilas). Por exemplo, na figura abaixo, que mostra um trecho do mapa da Costa do Mosquito, há apenas um percurso entre as vilas A e C, passando pelas vilas B, G e D. Uma tarifa fixa de M$ 1,00 é cobrada por cada viagem entre vilas vizinhas; assim, para uma viagem de A a C o usuário gasta M$ 4,00.

Devido a um inesperado surto de dengue, o Ministério da Saúde da Costa do Mosquito resolveu montar um Posto de Vacinação. Para evitar que habitantes gastem muito dinheiro para se deslocar até a vila onde ficará o Posto de Vacinação, o Ministério da Saúde decidiu que este deverá ser instalado em uma vila de forma que o gasto com transporte até o Posto, para os habitantes que gastarem mais, seja o menor possível (para o caso da figura abaixo a vila escolhida seria G).

Tarefa

Sua tarefa é escrever um programa que determine uma vila onde deve ser instalado o Posto de Vacinação. Esta vila deve ser tal que o custo com transporte, para os habitantes que tiverem maior custo, seja o menor possível. Note que devido à característica peculiar do sistema viário, ou haverá uma única vila que satisfaz essa restrição, ou haverá duas vilas que a satisfazem. No caso de existirem duas vilas apropriadas, qualquer uma delas serve como solução.

Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro N que indica a quantidade de vilas do país. As vilas são numeradas de 1 a N. As N-1 linhas seguintes contêm, cada uma, dois inteiros positivos X e Y que indicam que a vila X tem um caminho que a liga diretamente com a vila Y, sem passar por outras vilas. O final da entrada é indicado por N = 0.

Exemplo de Entrada

```
2
1 2
7
1 2
2 5
7 4
7 2
4 6
3 4
1
0 0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter o número da vila na qual deve ser instalado o Posto de Vacinação. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
1
Teste 2
7
Teste 3
1
```

(esta saída corresponde ao exemplo de entrada acima)

Restrições

$$0 \leq N \leq 100$$

(N = 0 apenas para indicar o final da entrada)

Exercício 2.11: Leitor binário

Escreva um programa em Java que permita escolher um arquivo, abri-lo em formato binário e apresentar em uma janela o seu conteúdo de forma tabular, com uma coluna de índice e colunas com os valores dos bytes do arquivo, em formato hexadecimal, seguindo as definições apresentadas na tabela a seguir:

Indice	Conteúdo do arquivo		
0000	00 01 02 03	FF 00 FF 00	A2 3F 19 33
0012	00 01 02 03	FF 00 FF 00	A2 3F 19 33
0024	00 01 02 03	FF 00 FF 00	A2 3F 19 33
0036	00 01 02 03	FF 00 FF 00	A2 3F 19 33

Exercício 2.12: Gravador binário

Escreva um programa que permita ao usuário escrever uma sequencia de bytes, podendo ele escolher entre digitar em formato decimal, hexadecimal ou binário, e em seguida os valores devem ser gravados em um arquivo, respeitando os valores digitados pelo usuário em formato digital. Deve ser possível também escolher gravar o arquivo em formato texto, de maneira que o valor de cada byte digitado seja separado por um espaço no arquivo, e seja sempre gravado em formato hexadecimal.

Uma sugestão para a interface do programa é a utilização de dados tabulados, como apresentado na tabela a seguir:

Índice	Conteúdo do arquivo	
	Binário	
0000	00000000 01010101 10101010 11110000	00000000 01010101 10101010 11110000
0008	00000000 01010101 10101010 11110000	00000000 01010101 10101010 11110000
0016	00000000 01010101 10101010 11110000	00000000 01010101 10101010 11110000
0024	00000000 01010101 10101010 11110000	00000000 01010101 10101010 11110000

Exercício 2.13: Editor binário

Escreva um programa que permita ao usuário escolher um arquivo, abri-lo em formato binário e apresentar em sua janela o conteúdo de forma tabular, sendo possível escolher entre a apresentação dos dados em formato binário, decimal, octal ou hexadecimal. Além de apresentar as informações, o programa deve permitir ao usuário alterar o conteúdo e

em seguida salvar as alterações, no mesmo arquivo ou em outro a sua escolha. Também deve ser possível escolher gravar o conteúdo do arquivo em formato textual, conforme o apresentado em sua janela.

Na tabela a seguir é apresentada uma sugestão para a apresentação do conteúdo do arquivo na janela do programa:

Indice	Conteúdo do arquivo		
	Decimal		
0000	000 001 002 003	255 00 125 00	075 100 019 241
0012	000 001 002 003	255 00 123 00	023 100 019 250
0024	000 001 002 003	250 00 255 00	045 100 019 033
0036	000 001 002 003	200 00 255 00	125 200 019 033

Exercício 2.14: Arquivo padrão Bitmap

Escreva um programa em java que contenha uma classe capaz de ler um arquivo do tipo Bitmap (.BMP) do disco, uma segunda classe que seja capaz de apresentar o conteúdo do arquivo em um painel, para que o usuário possa visualizar a imagem contida no arquivo.
Descrição do formato de arquivo BMP:

Todo arquivo BMP está dividido em 3 ou 4 partes, que são:

Cabeçalho de arquivo Contém a assinatura BM e informações sobre o tamanho e layout do arquivo BMP (disposição dos dados dentro do arquivo)

Cabeçalho de mapa de bits Contém as informações da imagem contida no arquivo. Define as dimensões, tipo de compressão (se houver) e informações sobre as cores da imagem.

Paleta ou mapa de cores (opcional) Somente estará presente em arquivos de imagens que usam 16 ou 256 cores (4 e 8 bits/pixel). Nas demais, em seu lugar, vem diretamente a parte seguinte: área de dados da imagem.

Área de dados da imagem contida no arquivo Dados que permitem a exibição da imagem propriamente dita, o dados dos pixels a serem exibidos. Podem ser com ou sem compressão.

Existem dentro da estrutura do BMP alguns campos ditos ”Reservados”, destinados a uso futuro, que sempre devem ser setados com ZERO. Outros dados são sempre idênticos ou fornecidos mais de uma vez no arquivo, para possível re-checagem e desvio de possíveis erros.

Cabeçalho de arquivo

Informações do arquivo (Tamanho: 14 bytes)

Campo	Tamanho em Bytes	Descrição
BfType	2	Assinatura do arquivo: os caracteres ASCII "BM"ou $(424D)_h$. É a identificação de ser realmente BMP.
BfSize	4	Tamanho do arquivo em Bytes.
BfReser1	2	Campo reservado 1; deve ser ZERO.
BfReser2	2	Campo reservado 2; deve ser ZERO.
BfOffSetBits	4	Especifica o deslocamento, em bytes, para o início da área de dados da imagem, a partir do início deste cabeçalho. Se a imagem usa paleta, este campo tem tamanho=14+40+(4 x NumeroDeCores). Se a imagem for true color, este campo tem tamanho=14+40=54.

Cabeçalho de mapa de bits

Informações da imagem (Tamanho: 40 bytes)

Campo	Tamanho em Bytes	Descrição
BiSize	4	Tamanho deste cabeçalho (40 bytes). Sempre $(28)_h$.
BiWidth	4	Largura da imagem em pixels.
BiHeight	4	Altura da imagem em pixels.
BiPlanes	2	Número de planos de imagem. Sempre 1.
BiBitCount	2	Quantidade de Bits por pixel (1,4,8,24,32). Este campo indica, indiretamente, ainda o número máximo de cores, que é 2 Bits por pixel.

Continua na próxima página.

Continuação da página anterior

Campo	Tamanho em Bytes	Descrição
BiCompress	4	Compressão usada. Pode ser: • 0 = BI_RGB → sem compressão • 1 = BI_RLE8 → compressão RLE 8 bits • 2 = BI_RLE4 → compressão RLE 4 bits
BiSizeImag	4	Tamanho da imagem (dados) em byte • Se arquivo sem compressão, este campo pode ser ZERO. • Se imagem em true color, será Tamanho do arquivo (Bfsize) menos deslocamento (BfOffSetBits)
BiXPPMeter	4	Resolução horizontal em pixels por metro.
BiYPPMeter	4	Resolução vertical em pixels por metro.
BiClrUsed	4	Número de cores usadas na imagem. Quando ZERO indica o uso do máximo de cores possível pela quantidade de Bits por pixel , que é 2 Bits por pixel.
BiClrImpor	4	Número de cores importantes (realmente usadas) na imagem. Por exemplo das 256 cores, apenas 200 são efetivamente usadas. Se todas são importantes pode ser ZERO. É útil quando for exibir uma imagem em 1 dispositivo que suporte menos cores que a imagem possui.

Paleta de cores

definição da tabela de cores (Tamanho: 4 bytes x Número de Cores)

Campo	Tamanho em Bytes	Descrição
Blue	1	Intensidade de Azul. De 0 a 255.
Green	1	Intensidade de Verde. De 0 a 255.
Red	1	Intensidade de Vermelho. De 0 a 255.
Reservado	1	Campo reservado deve ser ZERO sempre.

A tabela de cores só está presente em imagens com 256 cores ou menos. Para os demais tipos de BMP, vem em seu lugar a área de dados. No BMP a cor é representada de forma diferenciada dos demais formatos de arquivos de imagens.

A paleta é um vetor de bytes da estrutura RGBA, representando a intensidade de cada cor, através de 1 byte. Imagens de 8 bits por pixel, com no máximo 256 cores terão 256 posições na paleta; da mesma forma que imagens de 4 bits por pixel terão 16 posições e imagens de 1 bit por pixel terão 2 posições na paleta.

Área de dados da imagem:

Cor que cada pixel deve ser ligado ou esses dados comprimidos - Tamanho: campo BiSizeImg, do cabeçalho de informações da imagem.

Esta área do arquivo de imagens varia, conforme existência ou não de compressão. Para imagens sem compressão, os dados são armazenados em uma ordem sequencial, dentro do arquivo, que corresponde a posições na tela de vídeo. O primeiro pixel refere-se a posição inferior esquerda. O último pixel refere-se a posição superior direita. Como a orientação usual de um sistema de coordenadas cartesiano.

O valor lido nessa área de dados, se sem compressão, refere-se a cor do pixel de acordo com a tabela de cores (paleta).

Há, entretanto, a restrição de que cada linha deva ter N bytes, sendo N um número divisível por 4. Caso contrário, o BMP deve ser preenchido com bytes não válidos. Por exemplo, se a imagem tem 1 x 100 pixels em 8 bits/pixel, o BMP teria 1 byte válido em cada linha e mais 3 bytes que não tem qualquer significado.

No BMP monocromático, cada valor lido corresponde a uma entrada na paleta de cores. Se o bit for ZERO a cor é a da paleta[0]; caso contrário a cor será a da paleta[1]. No BMP de 16 cores, cada 4 bits (meio byte) correspondem a uma entrada na paleta de cores. Por exemplo se o primeiro byte contiver (1F)h, o primeiro pixel tem a cor da palera[0] e o segundo pixel a cor da paleta[15].

No BMP de 256 cores, cada byte (8 bits) correspondem a uma entrada na paleta de cores. Se o primeiro byte da área de dados contém (1F)h o primeiro pixel tem a cor da paleta[31].

No BMP true color (24 bits) cada sequência de 3 bytes correspondem a uma sequência Blue.Green.Red , isso é a composição da cor do pixel diretamente (não tendo neste caso paleta de cores). No true color de 32 bits, é a combinação de 4 bytes que estabelecerá a cor do pixel.

Exercício 2.15: Controle de estoque

Faça um programa que simule um controle de estoque de uma loja, onde cada produto, representado por meio de um registro, possui um identificador inteiro, nome, quantidade, preço de custo unitário e preço de venda unitário. O programa deve permitir a inclusão e remoção de novos produtos, cadastro de venda (que possui um identificador, a data da venda, a quantidade vendida, o preço unitário e o desconto concedido), cadastro de compra (que possui um identificador, a data da compra do produto, a quantidade comprada e o valor unitário pago), consulta de produtos por nome, alteração de registros, geração de relatórios (ex: qual vendeu mais, qual tem maior estoque, produtos cujo estoque estejam abaixo de X unidades, qual a quantidade vendida em determinado período, etc.).

Todos os registros devem ser armazenados sequencialmente em arquivos binários. Deve ser criado um arquivo para cada tipo de informação, ou seja, um arquivo com os registros de produtos, um arquivo com os registros de vendas e um arquivo com os registros de compras. Como as informações no arquivo não serão organizadas, crie arquivos auxiliares com os índices para que as informações sejam acessadas com maior agilidade no caso de consultas ordenadas alfabeticamente ou por valor de algum dos campos do registro.

O programa deve disponibilizar uma interface gráfica ao usuário (GUI) com opções para acessar as telas de cadastro de produtos, de vendas e de compras de produtos além de possibilitar a visualização dos relatórios.

Exercício 2.16: Arquivo de texto tabular

Tendo-se um arquivo texto, que possui seus dados (numéricos) dispostos em colunas de valores inteiros, faça um programa que imprima na tela apenas o valor de uma coluna especificada pelo usuário. O programa deve possibilitar também que a coluna selecionada seja gravada em outro arquivo, de forma textual ou binária, a escolha do usuário.

Exercício 2.17: Notas da turma

Escreva um programa para ler o conteúdo de um arquivo contendo as notas obtidas pelos alunos de uma turma. O programa deve exibir na tela o valor da nota mínima, valor da nota máxima e o total de alunos aprovados e reprovados (considere que o aluno é aprovado se a nota for maior ou igual a 7.0). O formato do arquivo é o seguinte:

```
7.5
8.4
9.1
4.0
5.7
4.3
6.2
```

Exercício 2.18: Alunos aprovados e reprovados

Escreva um programa para ler o conteúdo do arquivo "notas.txt" contendo as 3 notas obtidas por cada aluno em uma disciplina. O programa deve gerar dois novos arquivos: o arquivo "aprovados.txt" com as notas finais dos alunos aprovados e o arquivo "reprovados.txt" com as notas finais dos alunos reprovados (considere que o aluno é aprovado se a nota for maior ou igual a 7.0). O formato do arquivo é o seguinte:

```
Alberto 3.5 5.5 7.5
Claudia 8.0 4.5 8.4
Denis   9.1 8.5 10.0
Gloria  4.0 4.5 6.2
Mario   5.3 7.5 5.7
Pedro   4.5 8.0 7.3
Renata  7.5 8.5 6.2
```

Exercício 2.19: Retângulos, triângulos e círculos

Considere um arquivo texto que descreve um conjunto de retângulos, triângulos e círculos. Cada linha do arquivo contém a descrição de uma figura. O primeiro número não branco da linha indica o tipo da figura: '1' para retângulo, '2' para triângulo e '3' para círculo. Esse número é seguido por valores reais: valores da base e da altura, no caso de retângulos e triângulos, e valor do raio, no caso de círculo. O arquivo pode conter eventuais linhas em branco. Um exemplo deste formato é mostrado abaixo.

```
1 7.5 4.5
2 3.3 8.4
3 9.1
3 4.0
1 5.7
3 5.2
2 2.0 6.2
```

Escreva um programa que leia o arquivo "entrada.txt", que contém as descrições das figuras no formato descrito acima, e imprima na tela o valor da média das áreas das figuras listadas no arquivo. Se não for possível abrir o arquivo, o programa deve ter como saída a mensagem "ERRO". Se não existir nenhuma figura no arquivo (arquivo existente, mas vazio), deve-se imprimir a mensagem "VAZIO".

Exercício 2.20: Atualizar notas da turma

Faça um programa para atualizar o arquivo de notas dos alunos de uma determinada disciplina. O arquivo "disciplina.dat" possui as seguintes informações:

Nome do Aluno	Número da matricula
String de comprimento 50	Inteiro

O arquivo "graus1.dat" possui as seguintes informações por aluno:

Número da matricula	G1
Inteiro	Real

Sabe-se que a turma possui, no máximo, 35 alunos.

No arquivo "graus2.dat", estão armazenadas as notas dos 3 testes que cada aluno fez para compor o seu grau G2. Para cada teste realizado por um aluno, haverá um registro com o seguinte formato:

Número da matricula	Nota do teste
Inteiro	Real

Independentemente do aluno ter feito ou não todos os testes, o seu grau G2 será a média aritmética dos testes. O programa deve criar um novo arquivo "notas.dat" com as notas dos 2 graus de cada aluno, conforme o exemplo abaixo:

Nome do aluno	Número da matricula	G1	G2
String	Inteiro	Real	Real

O programa deve realizar as verificações quanto a integridade dos dados, e caso exista algum problema, o mesmo deve ser informado ao usuário, e o processamento deve continuar normalmente.

Exercício 2.21: Questões

1. Em que casos é vantajoso trabalhar com arquivos?
2. Qual a sequência de passos que devem ser utilizados para permitir trabalhar com arquivos?
3. Quais são os tipos de arquivo tratáveis pela linguagem Java, suas características e quais os possíveis modos de operação sobre os mesmos?

Lista 3: Acesso a banco de dados relacional

Os scripts SQL, para o servidor MySQL, apresentados a seguir devem ser utilizados na solução dos exercícios.

Books:

```
1  CREATE DATABASE IF NOT EXISTS books;
2
3  USE books;
4
5  DROP TABLE IF EXISTS authorISBN;
6  DROP TABLE IF EXISTS titles;
7  DROP TABLE IF EXISTS authors;
8  DROP TABLE IF EXISTS publishers;
9
10 CREATE TABLE publishers (
11    publisherID INT NOT NULL AUTO_INCREMENT,
12    publisherName varchar (30) NOT NULL,
13    PRIMARY KEY (publisherID)
14 ) TYPE=INNODB;
15
16 CREATE TABLE authors (
17    authorID INT NOT NULL AUTO_INCREMENT,
18    firstName varchar (20) NOT NULL,
19    lastName varchar (30) NOT NULL,
20    PRIMARY KEY (authorID)
21 ) TYPE=INNODB;
22
23 CREATE TABLE titles (
24    isbn varchar (20) NOT NULL,
25    title varchar (100) NOT NULL,
26    editionNumber INT NOT NULL,
27    copyright varchar (4) NOT NULL,
28    publisherID INT NOT NULL,
29    imageFile varchar (20) NOT NULL,
30    price REAL NOT NULL,
31    PRIMARY KEY (isbn),
32    INDEX (publisherID),
33    FOREIGN KEY (publisherID) REFERENCES publishers(publisherID)
34 ) TYPE=INNODB;
35
36 CREATE TABLE authorISBN (
37    authorID INT NOT NULL,
38    isbn varchar (20) NOT NULL,
39    INDEX (authorID),
40    FOREIGN KEY (authorID) REFERENCES authors (authorID),
```

```
41     INDEX (isbn),
42     FOREIGN KEY (isbn) REFERENCES titles (isbn)
43  ) TYPE=INNODB;
44
45  insert into publishers (publisherName) values ('Prentice Hall');
46  insert into publishers (publisherName) values ('Prentice Hall ↩
       PTG');
47
48  insert into authors (firstName,lastName) values ('Harvey','↩
       Deitel');
49  insert into authors (firstName,lastName) values ('Paul','Deitel'↩
       );
50  insert into authors (firstName,lastName) values ('Tem','Nieto');
51  insert into authors (firstName,lastName) values ('Sean','Santry'↩
       );
52
53  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0131426443','C How to ↩
       Program',4,'2004',1,'chtp4.jpg',85.00);
54  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130895725','C How to ↩
       Program',3,'2001',1,'chtp3.jpg',69.95);
55  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0132261197','C How to ↩
       Program',2,'1994',1,'chtp2.jpg',49.95);
56  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130384747','C++ How to↩
        Program',4,'2003',1,'cpphtp4.jpg',85.00);
57  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130895717','C++ How to↩
        Program',3,'2001',1,'cpphtp3.jpg',69.95);
58  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0135289106','C++ How to↩
        Program',2,'1998',1,'cpphtp2.jpg',49.95);
59  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('013100252X','The ↩
       Complete C++ Training Course',4,'2003',2,'cppctc4.jpg'↩
       ,109.99);
60  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0139163050','The ↩
       Complete C++ Training Course',3,'2001',2,'cppctc3.jpg'↩
       ,109.95);
61  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('013028419x','e-Business↩
        and e-Commerce How to Program',1,'2001',1,'ebechtp1.jpg'↩
       ,69.95);
```

```
62 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0131450913','Internet ↩
     and World Wide Web How to Program',3,'2004',1,'iw3htp3.jpg'↩
     ,85.00);
63 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130308978','Internet ↩
     and World Wide Web How to Program',2,'2002',1,'iw3htp2.jpg'↩
     ,69.95);
64 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130161438','Internet ↩
     and World Wide Web How to Program',1,'2000',1,'iw3htp1.jpg'↩
     ,69.95);
65 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130856118','The ↩
     Complete Internet and World Wide Web Programming Training ↩
     Course',1,'2000',2,'iw3ctc1.jpg',109.95);
66 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0131483986','Java How ↩
     to Program',6,'2005',1,'jhtp6.jpg',85.00);
67 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0131016210','Java How ↩
     to Program',5,'2003',1,'jhtp5.jpg',85.00);
68 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130341517','Java How ↩
     to Program',4,'2002',1,'jhtp4.jpg',85.00);
69 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130125075','Java How ↩
     to Program (Java 2)',3,'2000',1,'jhtp3.jpg',69.95);
70 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0138993947','Java How ↩
     to Program (Java 1.1)',2,'1998',1,'jhtp2.jpg',49.95);
71 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130852473','The ↩
     Complete Java 2 Training Course',3,'2000',2,'javactc3.jpg'↩
     ,109.95);
72 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130829277','The ↩
     Complete Java Training Course (Java 1.1)',2,'1998',2,'↩
     javactc2.jpg',99.95);
73 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0134569555','Visual ↩
     Basic 6 How to Program',1,'1999',1,'vbhtp1.jpg',69.95);
74 insert into titles (isbn,title,editionNumber,copyright,↩
     publisherID,imageFile,price) values ('0130829293','The ↩
     Complete Visual Basic 6 Training Course',1,'1999',2,'vbctc1.↩
     jpg',109.95);
```

```
75  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130284173','XML How to↩
        Program',1,'2001',1,'xmlhtp1.jpg',69.95);
76  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130284181','Perl How ↩
       to Program',1,'2001',1,'perlhtp1.jpg',69.95);
77  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130895601','Advanced ↩
       Java 2 Platform How to Program',1,'2002',1,'advjhtp1.jpg'↩
       ,69.95);
78  insert into titles (isbn,title,editionNumber,copyright,↩
       publisherID,imageFile,price) values ('0130461342','Java Web ↩
       Services for Experienced Programmers',1,'2003',1,'jwsfep1.jpg↩
       ',54.99);
79
80  insert into authorISBN (authorID,isbn) values (1,'0130895725');
81  insert into authorISBN (authorID,isbn) values (1,'0132261197');
82  insert into authorISBN (authorID,isbn) values (1,'0130895717');
83  insert into authorISBN (authorID,isbn) values (1,'0135289106');
84  insert into authorISBN (authorID,isbn) values (1,'0139163050');
85  insert into authorISBN (authorID,isbn) values (1,'013028419x');
86  insert into authorISBN (authorID,isbn) values (1,'0130161438');
87  insert into authorISBN (authorID,isbn) values (1,'0130856118');
88  insert into authorISBN (authorID,isbn) values (1,'0130125075');
89  insert into authorISBN (authorID,isbn) values (1,'0138993947');
90  insert into authorISBN (authorID,isbn) values (1,'0130852473');
91  insert into authorISBN (authorID,isbn) values (1,'0130829277');
92  insert into authorISBN (authorID,isbn) values (1,'0134569555');
93  insert into authorISBN (authorID,isbn) values (1,'0130829293');
94  insert into authorISBN (authorID,isbn) values (1,'0130284173');
95  insert into authorISBN (authorID,isbn) values (1,'0130284181');
96  insert into authorISBN (authorID,isbn) values (1,'0130895601');
97  insert into authorISBN (authorID,isbn) values (1,'0130384747');
98  insert into authorISBN (authorID,isbn) values (1,'0131426443');
99  insert into authorISBN (authorID,isbn) values (1,'013100252X');
100 insert into authorISBN (authorID,isbn) values (1,'0131450913');
101 insert into authorISBN (authorID,isbn) values (1,'0130308978');
102 insert into authorISBN (authorID,isbn) values (1,'0131483986');
103 insert into authorISBN (authorID,isbn) values (1,'0131016210');
104 insert into authorISBN (authorID,isbn) values (1,'0130341517');
105 insert into authorISBN (authorID,isbn) values (1,'0131450913');
106 insert into authorISBN (authorID,isbn) values (1,'0130308978');
107
108 insert into authorISBN (authorID,isbn) values (2,'0130895725');
109 insert into authorISBN (authorID,isbn) values (2,'0132261197');
110 insert into authorISBN (authorID,isbn) values (2,'0130895717');
```

```
111 insert into authorISBN (authorID,isbn) values (2,'0135289106');
112 insert into authorISBN (authorID,isbn) values (2,'0139163050');
113 insert into authorISBN (authorID,isbn) values (2,'013028419x');
114 insert into authorISBN (authorID,isbn) values (2,'0130161438');
115 insert into authorISBN (authorID,isbn) values (2,'0130856118');
116 insert into authorISBN (authorID,isbn) values (2,'0130125075');
117 insert into authorISBN (authorID,isbn) values (2,'0138993947');
118 insert into authorISBN (authorID,isbn) values (2,'0130852473');
119 insert into authorISBN (authorID,isbn) values (2,'0130829277');
120 insert into authorISBN (authorID,isbn) values (2,'0134569555');
121 insert into authorISBN (authorID,isbn) values (2,'0130829293');
122 insert into authorISBN (authorID,isbn) values (2,'0130284173');
123 insert into authorISBN (authorID,isbn) values (2,'0130284181');
124 insert into authorISBN (authorID,isbn) values (2,'0130895601');
125 insert into authorISBN (authorID,isbn) values (2,'0130384747');
126 insert into authorISBN (authorID,isbn) values (2,'0131426443');
127 insert into authorISBN (authorID,isbn) values (2,'013100252X');
128 insert into authorISBN (authorID,isbn) values (2,'0131450913');
129 insert into authorISBN (authorID,isbn) values (2,'0130308978');
130 insert into authorISBN (authorID,isbn) values (2,'0131483986');
131 insert into authorISBN (authorID,isbn) values (2,'0131016210');
132 insert into authorISBN (authorID,isbn) values (2,'0130341517');
133 insert into authorISBN (authorID,isbn) values (2,'0131450913');
134 insert into authorISBN (authorID,isbn) values (2,'0130308978');
135
136 insert into authorISBN (authorID,isbn) values (3,'013028419x');
137 insert into authorISBN (authorID,isbn) values (3,'0130161438');
138 insert into authorISBN (authorID,isbn) values (3,'0130856118');
139 insert into authorISBN (authorID,isbn) values (3,'0134569555');
140 insert into authorISBN (authorID,isbn) values (3,'0130829293');
141 insert into authorISBN (authorID,isbn) values (3,'0130284173');
142 insert into authorISBN (authorID,isbn) values (3,'0130284181');
143
144 insert into authorISBN (authorID,isbn) values (4,'0130895601');
```

Exercício 3.1: Preencha as lacunas

Preencha as lacunas em cada uma das afirmações:

1. A linguagem padrão internacional de banco de dados é _____.
2. Uma tabela em um banco de dados consiste em _____ e _____.
3. Os objetos de instrução retornam resultados de consulta de SQL como objetos _____.
4. O(A) _____ identifica unicamente cada linha em uma tabela.

5. A palavra-chave de SQL _____ é seguida pelos critérios de seleção que especificam as linhas a selecionar em uma consulta.
6. As palavra-chave de SQL _____ especificam a ordem em que linhas são classificadas em uma consulta.
7. Mesclar linhas de múltiplas tabelas de banco de dados é chamado de fazer _____ das tabelas.
8. Um(a) _____ é uma coleção organizada de dados.
9. Um(a) _____ é um conjunto de colunas cujos valores correspondem aos valores de chave primária de outra tabela.
10. Um objeto _____ é utilizado para obter uma `Connection` com um banco de dados.
11. A interface _____ ajuda a gerenciar a conexão entre um programa Java e um banco de dados.
12. Um objeto _____ é utilizado para submeter uma consulta a um banco de dados.
13. Ao contrário de um objeto `ResultSet`, os objetos _____ e _____ são roláveis e atualizáveis por padrão.
14. _____, um `RowSet` desconectado, armazena os dados em cache de um `ResultSet` na memória.

Exercício 3.2: Consultas para a base *books*

Considerando o banco de dados ***books***, defina um aplicativo de consulta completo, fornecendo as seguintes consultas predefinidas:

1. Selecione todos os autores da tabela `authors`.
2. Selecione todos os editores da tabela `publishers`.
3. Selecione um autor específico e liste todos os livros para este autor. Inclua o título, ano e ISBN. Ordene as informações alfabeticamente pelo nome e sobrenome do autor.
4. Selecione um editor específico e liste todos os livros publicados por esse editor. Inclua o título, ano e ISBN. Ordene as informações alfabeticamente por título.

Exiba um `JComboBox` com nomes apropriados para cada consulta predefinida.

Exercício 3.3: Inserções para a base *books*

Considerando o banco de dados ***books***, defina um aplicativo de inserção, fornecendo as seguintes telas de cadastro:

1. Cadastro de autores, para a tabela `authors`.
2. Cadastro de editores, para a tabela `publishers`.
3. Cadastro de títulos, para a tabela `titles`.

Exiba um `JComboBox` com os nomes dos autores no momento de cadastrar os títulos, com o objetivo de vincular os registros.

Exercício 3.4: Remoções para a base *books*

Considerando o banco de dados ***books***, defina um aplicativo de remoção de registros, fornecendo as seguintes opções:

1. Remover um autor da tabela `authors`, certificando-se de que o mesmo não possui títulos associados.
2. Remover um autor da tabela `authors`, juntamente com todos os títulos associados.
3. Remover um editor da tabela `publishers`, certificando-se de que o mesmo não possui títulos associados.
4. Remover um editor da tabela `publishers`, juntamente com todos os títulos associados.
5. Remover um título da tabela `titles`.

Exiba um `JComboBox` com nomes apropriados para cada caso.

Exercício 3.5: Verificações para a base *books*

Considerando o banco de dados ***books***, defina um aplicativo de verificação para averiguar se os relacionamentos entre os registros estão corretos, e forneça uma lista com os registros órfãos. O aplicativo deve solicitar ao usuário se estes registros devem ser removidos da base de dados, em caso de remoção uma opção de salvar os registros em um arquivo de texto deve ser fornecida ao usuário.

Exercício 3.6: Copiar uma base de dados para arquivo

Considerando o banco de dados **`books`**, defina um aplicativo que crie um arquivo de texto com as informações correspondentes a cada um dos títulos cadastradas na base de dados, incluindo os dados dos autores e editores das obras.

Exercício 3.7: Controle de estoque

Crie uma base de dados que corresponde a um controle de estoque de uma loja, onde cada produto, possui um identificador inteiro, nome, quantidade, preço de custo unitário e preço de venda unitário. O programa deve permitir a inclusão e remoção de novos produtos, cadastro de venda (que possui um identificador, a data da venda, a quantidade vendida, o preço unitário e o desconto concedido), cadastro de compra (que possui um identificador, a data da compra do produto, a quantidade comprada e o valor unitário pago), consulta de produtos por nome, alteração de registros, geração de relatórios (ex: qual vendeu mais, qual tem maior estoque, produtos cujo estoque estejam abaixo de X unidades, qual a quantidade vendida em determinado período, etc.).

O programa deve disponibilizar uma interface gráfica ao usuário com opções para acessar as telas de cadastro de produtos, de vendas e de compras de produtos além de possibilitar a remoção e a alteração dos registros. Além das telas de cadastro, alteração e remoção a interface deve possibilitar também a visualização dos relatórios.

Lista 4: Genéricos

Exercício 4.1: Verdadeiro ou falso

Determine se cada sentença é verdadeira ou falsa. Se falsa, explique porque.

1. Um método genérico não pode ter o mesmo nome de um método não genérico.
2. Todas as declarações de métodos genéricos têm uma seção de de parâmetros de tipo que precede imediatamente o nome do método.
3. Um método genérico pode ser sobrecarregado por outro método genérico com o mesmo nome do método, mas diferentes parâmetros de método.
4. Um parâmetro de tipo pode ser declarado somente uma vez na seção de parâmetros de tipo, mas pode aparecer mais de uma vez na lista de parâmetros do método.
5. Os nomes dos parâmetros de tipo entre diferentes métodos genéricos devem ser únicos.
6. O escopo de um parâmetro de tipo da classe genérico é a classe inteira, exceto seus membros *static*.

Exercício 4.2: Preencha as lacunas

Preencha as lacunas em cada uma das seguintes afirmações:

1. _____ e _____ permitem que os programadores especifiquem, com uma única declaração de método, um conjunto de métodos relacionados ou com uma única declaração de classe, um conjunto de tipos relacionados, respectivamente.
2. Uma seção de parâmetro de tipo é delimitada por _____.
3. Os(As) _____ de um método genérico podem ser utilizados(as) para especificar os tipos dos argumentos do método, para especificar o tipo de retorno do método e para declarar variáveis dentro do método.
4. A instrução "*Stack objectStack = new Stack();*" indica que *objectStack* armazena _____.
5. Em uma declaração de uma classe genérica, o nome da classe é seguido por um(a) _____.
6. A sintaxe _____ especifica que o limite superior de um curinga é o tipo E.

Exercício 4.3: Questões

1. Explique o uso da seguinte notação em um programa Java:

   `public class Array <T>{}`

2. Como os métodos genéricos podem ser sobrecarregados? O compilador realiza um processo de correspondência para determinar qual método chamar quando um método é invocado. Sob quais circunstâncias uma tentativa de fazer uma correspondência resulta em um erro em tempo de compilação?

3. Explique por que um programa Java utilizaria a instrução:

   `ArrayList <Emploee>workerList = new ArrayList<Emploee>();`

Exercício 4.4: Implementar selectionSort

Escreva um método genérico *selectionSort* com base no programa de classificação listado abaixo. Escreva um programa de teste que insere, classifica e gera saída de um *array Integer* e de um *array Float*.

```
1   import java.util.Random;
2
3   public class SelectionSort
4   {
5      private int[] data;
6      private static Random generator = new Random();
7
8      public SelectionSort(int size)
9      {
10        data = new int[size];
11        for(int i = 0; i < size; i++)
12           data[i] = 10 + generator.nextInt(90);
13     }
14
15     public void sort()
16     {
17        int smallest;
18        for(int i=0; i < data.length - 1; i++)
19        {
20           smallest = i;
21           for(int index = 1 + 1; index < data.length; index++)
22              if(data[index] < data[smallest])
23                 smallest = index;
24           swap(i, smallest);
25           printPass(i+1, smallest);
26        }
27     }
```

```
28
29          public void swap(int first, int second)
30          {
31             int temporary = data[first];
32             data[first] = data[second];
33             data[second] = temporary;
34          }
35
36          public void printPass(int pass, int index)
37          {
38             System.out.print(String.format("aftar pass %2d: ", pass↩
                  ));
39
40             for(int i=0; i < index; i++)
41                System.out.print(data[i] + " ");
42
43             System.out.print(data[index] + "* ");
44
45             for(int i = index + 1; i < data.length; i++)
46                System.out.print(data[i] + " ");
47
48             System.out.print("\n              ");
49
50             for(int j = 0; j < pass; j++)
51                System.out.print("--  ");
52             System.out.println("\n");
53          }
54
55          public String toString()
56          {
57             StringBuffer temporary = new StringBuffer();
58             for(int element: data)
59                temporary.append(element + " ");
60             temporary.append("\n");
61             return temporary.toString();
62          }
63       }
64
65       public class SelectionSortTest
66       {
67          public static void main(String[] args)
68          {
69             SelectionSort sortArray = new SelectionSort(10);
70
71             System.out.println("Unsorted array: ");
72             System.out.println(sortArray);
```

```
73
74          sortArray.sort();
75
76          System.out.println("Sorted array: ");
77          System.out.println(sortArray);
78       }
79    }
```

Exercício 4.5: Sobrecarga de método genérico

Sobrecarregue o método genérico *printArray* da classe listada abaixo, de modo que ele aceite dois argumentos adicionais de inteiros, *lowSubscript* e *highSubscript*. Uma chamada a esse método imprime somente a parte especificada do array.

Valide *lowSubscript* e *highSubscript*. Se estiver fora do intervalo ou se *highSubscript* for menor ou igual a *lowSubscript*, o método *printArray* sobrecarregado deve lançar uma *InvalidSubscriptException*; caso contrário,*printArray* deve retornar o número de elementos impresso.

Em seguida modifique o método *main* para praticar as duas versões de *printArray* nos arrays *inteerArray*, *doubleArray* e *characterArray*.

Teste todas as capacidades das duas versões de *printArray*.

```
1    public class GenericMethodTest
2    {
3       public static < E > void printArray( E[] inputArray )
4       {
5          for ( E element : inputArray )
6             System.out.printf( "%s ", element );
7          System.out.println();
8       }
9
10         public static void main( String args[] )
11         {
12            Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
13            Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, ↩
                  7.7 };
14            Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' ↩
                  };
15
16            System.out.println( "Array integerArray contains:" );
17            printArray( integerArray ); // passa um array de ↩
                  Integers
18            System.out.println( "\nArray doubleArray contains:" );
19            printArray( doubleArray ); // passa um array Doubles
20            System.out.println( "\nArray characterArray contains:" ↩
                  );
```

```
21          printArray( characterArray ); // passa um array de ↩
               Characters
22       }
23    }
```

Exercício 4.6: Árvore binária genérica

Converta as classes *TreeNode* e *Tree* apresentadas a seguir em classes genéricas. Para inserir um objeto em uma *Tree*, o objeto deve ser comparado aos objetos dos nós da *Tree* existentes. Por essa razão, as classes *TreeNode* e *Tree* devem especificar *Comparable<E>* como o limite superior de cada parâmetro de tipo de classe. Depois de modificar as classes *TreeNode* e *Tree*, escreva um aplicativo de teste que cria três objetos *Tree* - um que armazena *Integers*, um que armazena *Doubles* e outro que armazena *Strings*. Insira 10 valores em cada árvore. Em seguida, gere a saída dos percursos na pré-ordem ena pós-ordem para cada *Tree*.

```
1     class TreeNode
2     {
3        TreeNode leftNode;
4        int data;
5        TreeNode rightNode;
6
7        public TreeNode( int nodeData )
8        {
9           data = nodeData;
10          leftNode = rightNode = null;
11       }
12
13       public void insert( int insertValue )
14       {
15          if ( insertValue < data )
16          {
17             if ( leftNode == null )
18                leftNode = new TreeNode( insertValue );
19             else
20                leftNode.insert( insertValue );
21          }
22          else if ( insertValue > data )
23          {
24             if ( rightNode == null )
25                rightNode = new TreeNode( insertValue );
26             else
27                rightNode.insert( insertValue );
28          }
29       }
30    }
```

```
31
32    public class Tree
33    {
34       private TreeNode root;
35
36       public Tree()
37       {
38          root = null;
39       }
40
41       public void insertNode( int insertValue )
42       {
43          if ( root == null )
44             root = new TreeNode( insertValue );
45          else
46             root.insert( insertValue );
47       }
48
49       public void preorderTraversal()
50       {
51          preorderHelper( root );
52       }
53
54       private void preorderHelper( TreeNode node )
55       {
56          if ( node == null )
57             return;
58
59          System.out.printf( "%d ", node.data );
60          preorderHelper( node.leftNode );
61          preorderHelper( node.rightNode );
62       }
63
64       public void inorderTraversal()
65       {
66          inorderHelper( root );
67       }
68
69       private void inorderHelper( TreeNode node )
70       {
71          if ( node == null )
72             return;
73
74          inorderHelper( node.leftNode );
75          System.out.printf( "%d ", node.data );
76          inorderHelper( node.rightNode );
```

```
77        }
78
79        public void postorderTraversal()
80        {
81           postorderHelper( root );
82        }
83
84        private void postorderHelper( TreeNode node )
85        {
86           if ( node == null )
87              return;
88
89           postorderHelper( node.leftNode );
90           postorderHelper( node.rightNode );
91           System.out.printf( "%d ", node.data );
92        }
93     }
```

Exercício 4.7: Pares

Escreva uma classe genérica *Par* que tem dois parâmetros de tipo, P e S, cada um representando, respectivamente, o tipo do primeiro e o tipo do segundo elemento do par. Adicione os métodos *get* e *set* ao primeiro e ao segundo elemento do par.

Exercício 4.8: Desenho

Escreva um programa java que mostre ao usuário uma janela (GUI) contendo um painel de desenho, uma barra de mensagens na parte inferior da janela e uma barra de botões na parte superior da janela. Nesta barra, devem ser apresentadas opções para selecionar os modos de desenho correspondentes a pontos, retas, círculos ou elipses, retângulos e triângulos, um botão para gravar e outro para ler as informações de um arquivo utilizado para armazenar os dados do desenho. Deve ser definida uma interface que seja comum a todos os tipos de formas. Em memória os objetos de desenho devem ser organizados em uma lista encadeada genérica. Na barra de mensagens, deve ser mostrado um texto informando as características de cada desenho conforme o usuário os realiza com o auxílio do mouse. Ao clicar no botão para gravar, as informações de todos os desenhos feitos pelo usuário devem ser gravadas no arquivo, e ao clicar no botão para ler, as informações dos desenhos devem ser descartadas e substituídas pelas informações contidas no arquivo.

Exercício 4.9: Controle de estoque

Faça um programa que simule um controle de estoque de uma loja, onde cada produto, representado por meio de um registro, possui um identificador inteiro, nome, quantidade, preço de custo unitário e preço de venda unitário. O programa deve permitir a inclusão

e remoção de novos produtos, cadastro de venda (que possui um identificador, a data da venda, a quantidade vendida, o preço unitário e o desconto concedido), cadastro de compra (que possui um identificador, a data da compra do produto, a quantidade comprada e o valor unitário pago), consulta de produtos por nome, alteração de registros, geração de relatórios (ex: qual vendeu mais, qual tem maior estoque, produtos cujo estoque estejam abaixo de X unidades, qual a quantidade vendida em determinado período, etc.).

Todos os objetos de dados devem ser armazenados em memória utilizando listas encadeadas genéricas, quado necessário devem ser especificadas interfaces para a manipulação destas classes de dados.

Todos os registros devem ser armazenados sequencialmente em arquivos binários. Deve ser criado um arquivo para cada tipo de informação, ou seja, um arquivo com os registros de produtos, um arquivo com os registros de vendas e um arquivo com os registros de compras. Como as informações no arquivo não serão organizadas, crie arquivos auxiliares com os índices para que as informações sejam acessadas com maior agilidade no caso de consultas ordenadas alfabeticamente ou por valor de algum dos campos do registro.

O programa deve disponibilizar uma interface gráfica ao usuário (GUI) com opções para acessar as telas de cadastro de produtos, de vendas e de compras de produtos além de possibilitar a visualização dos relatórios.

Exercício 4.10: Método de ordenação

Na classe que implementa uma lista duplamente encadeada, listada abaixo, incluindo um método de ordenação, que ao ser chamado coloca os elementos da lista em ordem crescente ou decrescente. Escolha uma maneira de especificar o sentido de ordenação.

```
1   public interface Indice<P>
2   {
3      public P getProx();
4      public P getAnt();
5      public boolean hasNext();
6      public boolean hasPrev();
7   }
8
9   public class ListaDupEncadeada<E> {
10     private class No{
11        public No prox;
12        public No ant;
13        public E dado;
14        public No(E obj){
15           prox = null;
16           ant=null;
17           dado=obj;
18        }
19     }
20     private class IndiceLista implements Indice<E>{
```

```
21          public No noAtual;
22          @Override
23          public E getProx()
24          {
25             if(noAtual == null) return null;
26             E obj = noAtual.dado;
27             noAtual=noAtual.prox;
28             return obj;
29          }
30          @Override
31          public E getAnt() {
32             if(noAtual == null) return null;
33             E obj = noAtual.dado;
34             noAtual=noAtual.ant;
35             return obj;
36          }
37          @Override
38          public boolean hasNext() {
39             return noAtual.prox != null ? true : false;
40          }
41          @Override
42          public boolean hasPrev() {
43             return noAtual.ant != null ? true : false;
44          }
45       }
46       public Indice<E> getInicio()
47       {
48          IndiceLista indice = new IndiceLista();
49          indice.noAtual = inicio;
50          return indice;
51       }
52       public Indice<E> getFim()
53       {
54          IndiceLista indice = new IndiceLista();
55          indice.noAtual = fim;
56          return indice;
57       }
58
59       No inicio;
60       No fim;
61       int tamanho;
62
63       public ListaDupEncadeada() {
64          inicio = null;
65          fim = null;
66          tamanho = 0;
```

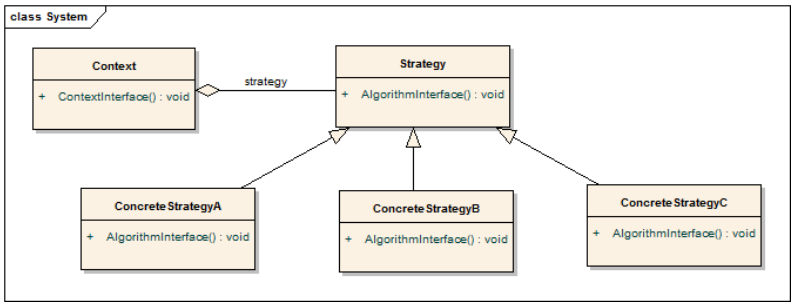
```
67        }
68
69
70        public int getTamanho(){
71           return tamanho;
72        }
73
74        public int insere(E obj, int pos){
75           No no = new No(obj);
76
77           if(inicio == null)//primeiro elemento - lista vazia
78           {
79              inicio = fim = no;
80           }
81           else // ja esistem elementos na lista
82           {
83              if(pos == 0)//inserir no inicio da lista
84              {
85                 no.prox = inicio;
86                 inicio.ant = no;
87                 inicio = no;
88              }
89              else if(pos >= tamanho-1)//inserir no final da lista
90              {
91                 no.ant = fim;
92                 fim.prox = no;
93                 fim = no;
94              }
95              else// inserir no meio da lista
96              {
97                 No aux = inicio;
98                 while(pos > 0)
99                 {
100                   aux = aux.prox;
101                   pos--;
102                }
103                // inserir na posicao de aux
104                no.prox = aux;
105                no.ant = aux.ant;
106                aux.ant.prox = no;
107                aux.ant = no;
108             }
109          }
110          tamanho++;
111          return tamanho;
112       }
```

```
113
114         public E remove(int pos)
115         {
116             E obj;
117             if(inicio == null)//se a lista esta vazia
118                 return null;
119             if(inicio == fim)//se existe apenas um elemento na ↩
                    lista
120             {
121                 obj = inicio.dado;
122                 inicio = fim = null;
123                 tamanho--;
124                 return obj;
125             }
126             if(pos==0)//remover o primeiro elemento da lista
127             {
128                 obj = inicio.dado;
129                 inicio = inicio.prox;
130             }
131             else if(pos >= tamanho-1)//remover o ultimo elemento da↩
                     lista
132             {
133                 obj = fim.dado;
134                 fim = fim.ant;
135             }
136             else //remover um elemento no meio da lista
137             {
138                 No aux = inicio;
139                 while(pos > 0){
140                     aux = aux.prox;
141                     pos--;
142                 }
143                 // remover o elemento aux
144                 obj = aux.dado;
145                 aux.ant.prox = aux.prox;
146                 aux.prox.ant = aux.ant;
147             }
148             tamanho--;
149             return obj;
150         }
151
152         public E consulta(int pos)
153         {
154             if(inicio == null)//se a lista esta vazia
155                 return null;
```

```
156             if(inicio == fim)//se existe apenas um elemento na ↩
                   lista
157                return inicio.dado;
158             if(pos==0)//consulta o primeiro elemento da lista
159                return inicio.dado;
160             else if(pos >= tamanho-1)//consulta o ultimo elemento ↩
                   da lista
161                return fim.dado;
162             else //consulta um elemento no meio da lista
163             {
164                No aux = inicio;
165                while(pos > 0){
166                   aux = aux.prox;
167                   pos--;
168                }
169                // consulta o elemento aux
170                return aux.dado;
171             }
172          }
173       }
```

Exercício 4.11: Padrão Strategy

Implementar um conjunto de algoritmos de ordenação a serem utilizados juntamente com a classe de lista encadeada apresentada no exercício 4.10. Deve ser especificada uma interface para acesso aos métodos de ordenação e cada método deve ser implementado em uma classe específica.



Lista 5: Programação concorrente - *Multithreading*

Exercício 5.1: Preencha as lacunas

Preencha as lacunas em cada uma das seguintes afirmações:

1. Uma thread entra no estado terminado quando _____.
2. Para pausar por um número designado de milisegundos e retornar a execução, uma thread deve chamar o método _____.
3. O método _____ da classe *Condition* move uma única thread no estado de espera de um objeto para o estado executável.
4. O método _____ da classe *Condition* move toda thread no estado de espera de um objeto para o estado executável.
5. Uma thread _____ entra no estado _____ quando ela completa sua tarefa ou, de outro modo, termina.
6. Uma thread executável pode entrar no estado _____ por um intervalo especificado de tempo.
7. No ível do sistema operacional, o estado executável realmente inclui dois estados separados, _____ e _____.
8. *Runnable*s são executadas utilizando uma classe que implementa a interface _____.
9. O método *ExecutorService.*_____ termina cada thread em uma interface *ExecutorService* logo que terminar de executar sua *Runnable* atual, se houver alguma.
10. Uma thread pode chamar o método _____ em um objeto *Condition* para liberar o *Lock* associado e colocar essa thread no estado _____.
11. Em um relacionamento _____, a parte _____ de um aplicativo gera dados e os armazena em um objeto compartilhado, e a parte _____ de um aplicativo lê os dados do objeto compartilhado.
12. Para minimizar a quantidade de tempo de espera de threads que compartilham recursos e operam nas mesmas velocidades médias, utilize um(a) _____.
13. A classe _____ implementa a interface *BlockingQueue* que utiliza um array.
14. A palavra-chave _____ indica que somente uma thread por vez deve executar em um objeto.

Exercício 5.2: Verdadeiro ou falso

Determine se cada uma das sentenças a seguir é verdadeira ou falsa. Se falsa, explique por quê.

1. Uma thread não é executável se estiver morta.
2. Em Java, uma thread de prioridade mais alta deve fazer preempção da thread de prioridade mais baixa.
3. Alguns sistemas operacionais utilizam o fracionamento de tempo com threads. Portanto, eles podem permitir às threads fazer preempção de threads de mesma prioridade.
4. Quando o quantum da thread expirar, a thread retorna ao estado de execução quando o sistema operacional a atribui a um processador.
5. Sem o fracionamento de tempo, toda thread em um conjunto de threads de igual prioridade executa até sua conclusão, antes que outras threads de igual prioridade tenham uma chance de executar.
6. o método *sleep* não consume tempo de processador enquanto uma thread dorme.
7. Utilizar um *Lock* garante que o impasse não pode ocorrer.
8. Uma vez que um *Lock* foi obtido por uma thread, o objeto *Lock* não permitirá que outra thread obtenha o bloqueio até que a primeira o libere.
9. Os componentes Swing são seguros para thread.

Exercício 5.3: Defina

Defina cada um dos seguintes termos:

1. thread
2. multithreading
3. estado executável
4. estado de espera sincronizada
5. agendamento preemptivo
6. interface *Runnable*
7. método *signal* da classe *Condition*
8. relacionamento de produtor/consumidor
9. quantum

Exercício 5.4: FirstThread

Considere os seguintes códigos:

```
1  public class FirstThread implements Runnable{
2     private String nome;
3     public FirstThread(String n){
4        nome = n;
5     }
6     public void run(){
7        System.out.println("Executou "+nome);
8     }
9     public static void main(String args[]){
10       FirstThread firstT = new FirstThread("MyFirstThread");
11       Thread t = new Thread(firstT);
12       t.start();
13    }
14 }
```

```
1  public class FirstThreadTest{
2     public static void main(String args[]){
3     int n = 10;
4     for(int i=0; i<n; i++){
5     FirstThread firstT = new FirstThread("MyFirstThread" + i);
6     Thread t = new Thread(firstT);
7     t.start();
8     }
9  }
```

```
1  public class SecondThread extends Thread{
2     public SecondThread(String n){
3        super(n); //opcional
4     }
5     public void run(){
6        System.out.println("Executou "+getName());
7     }
8     public static void main(String args[]){
9        SecondThread secT = new SecondThread("SecThread");
10       secT.start();
11    }
12 }
```

Faça as seguintes atividades:

1. Executar a thread listada na classe FirstThread.
2. Alterar a atividade anterior inserindo um comando sleep() no método run(), observando a ordem em que as threads iniciam e terminam a execução. Faça testes usando tempos fixos e também tempos variáveis (randômicos).

Exercício 5.5: Bolas

Elabore um programa que faça uma bola, cuja cor é definida aleatoriamente, rebater em um *JPanel*. A bola deve aparecer e começar a se mover com um evento *mousePressed*. Quando a bola atingir a borda do *JPanel*, ela deve rebater e continuar na direção oposta. A bola deve ser atualizada com uma interface *Runnable*.

Toda vez que o usuário clicar o mouse uma nova bola deve ser gerada. Um mínimo de 20 bolas devem ser permitidas.

Exercício 5.6: Classe Contador

Defina a classe *Contador* como uma subclasse de *Thread*, que imprime números de 0 a 10. Crie a classe *TesteContador* que deve definir o método *main* que cria e inicia a execução da thread *Contador*. Teste o resultado executando a classe *TesteContador*.

Altere as classes *Contador* e *TesteContador* de modo que a classe *Contador* seja definida como uma implementação da interface *Runnable*. Teste o resultado.

Agora altere o método *main* da classe *TesteContador* para criar duas ou mais threads *Contador* e inicialize a execução das mesmas.

Dicas:

Na última parte do exercício, onde é pedido para alterar o método *main* da classe *TesteContador*, você deve inicialmente criar as threads e em seguida inicializá-las, de modo a obter um melhor resultado na visualização da execução concorrente.

Como o laço do exemplo é apenas de 1 a 10 pode ser que não seja visualizado um interleaving ente as execuções das threads. Caso isto ocorra aumente o tamanho do laço.

Exercício 5.7: Mailbox

Defina uma classe *Mailbox* que tem um atributo *message* do tipo *String*. A classe *Mailbox* deve ter dois métodos: *storeMessage* e *retrieveMessage*. O método *storeMessage* recebe uma *String* e, se o mesmo tiver sido consumido *(message == null)*, armazena no atributo *message* do *Mailbox*. Caso contrário, quem chamou o método *storeMessage* deve esperar até que alguém consuma a mensagem (chamando o método *retrieveMessage*). De forma similar, o método *retrieveMessage* retorna o valor da mensagem, caso ela tenha sido produzida/armazenada *(message != null)*. Caso contrário quem chamou o método deve esperar até que alguém produza uma mensagem (chamando o método *storeMessage*).

Crie a classe *Producer*, que é uma thread e deve ter um atributo do tipo *Mailbox* e no seu método *run* deve ser definido um laço que executa o método *storeMessage* do *Mailbox*, armazenando mensagens no *Mailbox*.

Defina um a classe *Consumer*, que também é uma thread, e que deve consumir mensagens (chamando o método *retrieveMessage*) escritas no seu atributo do tipo *Mailbox*.

Crie uma classe de teste com um método *main* que cria um *Producer* e um *Consumer* que compartilham o mesmo *Mailbox* e iniciam a execução. Altere a classe de teste para criar mais de um produtor e/ou consumidor para o mesmo *Mailbox*.

Dicas:

Utilize os métodos *wait* e *notifiAll* para implementar os métodos da classe *Mailbox*.

Na última parte do exercício, onde é pedido para alterar a classe de teste, você deve adicionar um atributo de identificação (*String*) nas classes *Producer* e *Consumer*, de modo a permitir que sejam identificados tanto quem está consumindo uma mensagem, quanto quem esta produzindo a mesma. Para tal concatene o identificador dos objetos à mensagem a ser enviada (no caso do produtor) ou à mensagem a ser impressa (no caso do consumidor).

Exercício 5.8: Corrida de Sapos

Implemente uma Corrida de Sapos! Crie uma classe *Sapo* que herda de *Thread* com pelo menos este atributos: *distanciaPercorrida*, *distanciaPulo*, *velocidadePulo*. Crie os métodos que achar necessários.

Crie uma classe *CorridaSapos* com pelo menos estes atributos: *distanciaCorrida*, *numSapos*. Crie os métodos que achar necessários.

Crie uma classe de teste que executa a corrida de sapos, mostrando as posições dos sapos durante a corrida e faça um resumo da corrida ao seu final.

Exercício 5.9: Lista encadeada

Considerando o resultado obtido no exercício 4.10, altere a classe para que as operações com a lista sejam seguras, ou seja, a lista deve permitir que várias threads a utilizem como recurso compartilhado.

Escreva um programa de teste para esta classe de lista encadeada simulando um problema de produtor consumidor com pelo menos 5 threads ativas.

Lista 6: Redes - *TCP - UDP*

Exercício 6.1: Preencha as lacunas

Preencha as lacunas em cada uma das seguintes afirmações:

1. A exceção _____ ocorre quando um erro de entrada/saída ocorre ao fechar um socket.
2. A exceção _____ ocorre quando um nome de host indicado por um cliente não pode ser convertido em um endereço.
3. Se um construtor *DatagramSocket* não conseguir configurar um *DatagramSocket* adequadamente, uma exceção do tipo _____ ocorrerá.
4. Muitas classes para redes do Java estão contidas no pacote _____.
5. A classe _____ vincula o aplicativo a uma porta para transmissão de datagrama.
6. Um objeto da classe _____ contém um endereço IP.
7. O acrônimo URL significa _____.
8. O acrônimo URI significa _____.
9. O protocolo chave que forma a base da World Wide Web é _____.
10. O método *AppleContext.*_____ recebe um objeto URL como argumento e exibe em um navegador o recurso da World Wide Web associado com esse URL.
11. O método *getLocalHost* retorna um objeto _____ contendo o nome de host local do computador em que o programa está executando.
12. O método *MulticastSocket.*_____ faz uma assinatura de *MulticastSocket* para um grupo de multicast.
13. O construtor URL determina se seu argumento de string é um URL válido. Se for, o objeto URL é inicializado com essa localização. Se não, uma exceção _____ ocorre.

Exercício 6.2: Verdadeiro ou falso

Determine se cada uma das sentenças a seguir é verdadeira ou falsa. Se falsa, explique por quê.

1. O multicas transmite *DatagramPacket*s para cada host na Internet.

2. O UDP é um protocolo orientado para conexão.
3. Com sockets de fluxo um processo estabelece uma conexão com outro processo.
4. Um servidor espera conexões de um cliente em uma porta.
5. A transmissão de pacote de datagrama em uma rede é confiável - garante-se que os pacotes chegarão na sequência correta.
6. Por razões de segurança, muitos navegadores Web, como o Mozilla, permitem que applets Java façam processamento de arquivo apenas nas máquinas em que eles executam.
7. Os navegadores Web frequêntemente restringem um applet de modo que ele possa se comunicar apenas com a máquina de que originalmente foi descarregado.
8. Os endereços IP de 224.0.0.0 para 239.255.255.255 são reservados para multicast.

Exercício 6.3: Questões

1. Faça uma distinção entre serviços de rede sem conexão e orientados para conexão.
2. Como um cliente determina o nome de host do computador cliente?
3. Sob que circunstâncias uma *SocketException* será lançada?
4. Como um cliente pode obter uma linha de texto de um servidor?
5. Descreva como um cliente se conecta a um servidor.
6. Descreva como um servidor envia os dados a um cliente.
7. Descreva como preparar um servidor para receber uma solicitação de conexão baseada em fluxo a partir de um único cliente.
8. Como um servidor ouve conexões baseadas em fluxo de socket em uma porta?
9. O que determina quantas solicitações de conexão de clientes podem esperar em uma fila para se conectar a um servidor?
10. Que razões podem fazer com que um servidor recuse uma solicitação de conexão de um cliente?

Exercício 6.4: Solicitar arquivo

Utilize uma conexão de socket para permitir a um cliente especificar um nome de arquivo e fazer o servidor enviar o conteúdo do arquivo ou indicar que o arquivo não existe.

Exercício 6.5: Alterar arquivo

Utilize uma conexão de socket para permitir a um cliente especificar um nome de arquivo e fazer o servidor enviar o conteúdo do arquivo ou indicar que o arquivo não existe. Uma vez recebido o conteúdo do arquivo, o cliente ddeve permitir ao usuário modificar o conteúdo do arquivo e enviar o arquivo de volta ao servidor para armazenamento. O usuário pode editar o arquivo em uma *JTextArea*, então clicar em um botão identificado como ”Salvar alterações”para enviar o arquivo de volta ao servidor.

Exercício 6.6: Servidor multithred

```
1  public class Server extends JFrame
2  {
3     private JTextField enterField; // mensagem do usuario
4     private JTextArea displayArea; // exibe informacoes
5     private ObjectOutputStream output; // gera fluxo de saida
6     private ObjectInputStream input; // gera fluxo de entrada
7     private ServerSocket server; // socket de servidor
8     private Socket connection; // conexao com o cliente
9     private int counter = 1; // contador do numero de conexoes
10
11    // configura a GUI
12    public Server()
13    {
14       super( "Server" );
15       enterField = new JTextField(); // cria enterField
16       enterField.setEditable( false );
17       enterField.addActionListener(
18          new ActionListener()
19          {  // envia a mensagem ao cliente
20             public void actionPerformed( ActionEvent event )
21             {
22                sendData( event.getActionCommand() );
23                enterField.setText( "" );
24             }
25          }
26       );
27       add( enterField, BorderLayout.NORTH );
28       displayArea = new JTextArea(); // cria displayArea
29       add(new JScrollPane(displayArea), BorderLayout.CENTER);
30       setSize( 300, 150 ); // configura o tamanho da janela
31       setVisible( true ); // mostra a janela
32    }
33
34    // configura e executa o servidor
```

```
35    public void runServer()
36    {
37       try // configura o servidor; processa as conexoes
38       {
39          server = new ServerSocket( 12345, 100 );
40          while ( true )
41          {
42             try
43             {
44                waitForConnection(); // espera uma conexao
45                getStreams(); // obtem fluxos de entrada e saida
46                processConnection(); // processa a conexao
47             }
48             catch ( EOFException eofException )
49             {
50                displayMessage("\nServer terminated connection");
51             }
52             finally
53             {
54                closeConnection(); // fecha a conexao
55                counter++;
56             }
57          }
58       }
59       catch ( IOException ioException )
60       {
61          ioException.printStackTrace();
62       }
63    }
64
65    // espera que a conexao chegue e exibe informacoes
66    private void waitForConnection() throws IOException
67    {
68       displayMessage( "Waiting for connection\n" );
69       connection = server.accept();
70       displayMessage( "Connection " + counter + " received from:↩
             " + connection.getInetAddress().getHostName());
71    }
72
73    // obtem fluxos para enviar e receber dados
74    private void getStreams() throws IOException
75    {  // configura o fluxo de saida para objetos
76       output = new ObjectOutputStream( connection.↩
            getOutputStream() );
77       output.flush();
78       // configura o fluxo de entrada para objetos
```

```
79          input= new ObjectInputStream(connection.getInputStream());
80          displayMessage("\nGot I/O streams\n");
81       }
82
83       // processa a conexao com o cliente
84       private void processConnection() throws IOException
85       {
86          String message = "Connection successful";
87          sendData( message ); // mensagem de conexao bem-sucedida
88          setTextFieldEditable( true );
89          do // processa as mensagens enviadas pelo cliente
90          {
91             try // le e exibe a mensagem
92             {
93                message = ( String ) input.readObject();
94                displayMessage( "\n" + message );
95             }
96             catch ( ClassNotFoundException classNotFoundException )
97             {
98                displayMessage( "\nUnknown object type received" );
99             }
100         } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
101      }
102
103      // fecha os fluxos e o socket
104      private void closeConnection()
105      {
106         displayMessage( "\nTerminating connection\n" );
107         setTextFieldEditable( false ); // desativa enterField
108         try
109         {
110            output.close(); // fecha o fluxo de saida
111            input.close(); // fecha o fluxo de entrada
112            connection.close(); // fecha o socket
113         }
114         catch ( IOException ioException )
115         {
116            ioException.printStackTrace();
117         }
118      }
119
120      // envia a mensagem ao cliente
121      private void sendData( String message )
122      {
123         try // envia o objeto ao cliente
124         {
```

```
125             output.writeObject( "SERVER>>> " + message );
126             output.flush(); // esvazia a saida para o cliente
127             displayMessage( "\nSERVER>>> " + message );
128          }
129          catch ( IOException ioException )
130          {
131             displayArea.append( "\nError writing object" );
132          }
133       }
134
135       // manipula a displayArea na thread de despacho de eventos
136       private void displayMessage( final String messageToDisplay )
137       {
138          SwingUtilities.invokeLater(
139             new Runnable()
140             {
141                public void run() // atualiza a displayArea
142                {
143                   displayArea.append( messageToDisplay );
144                }
145             }
146          );
147       }
148
149       // manipula o enterField na thread de despacho de eventos
150       private void setTextFieldEditable( final boolean editable )
151       {
152          SwingUtilities.invokeLater(
153             new Runnable()
154             {
155                public void run()
156                {
157                   enterField.setEditable( editable );
158                }
159             }
160          );
161       }
162    }
```

Os servidores multiencadeados são bem populares hoje, especialmente por causa da utilização crescente de servidores multiprocessados. Modifique a classe apresentada acima para ser um servidor com multiplas threads. Então utilize vários aplicativos clientes e faça cada um deles se conectar ao servidor simultâneamente. Utilize um *ArrayList* para armazenar as threads de cliente.

Exercício 6.7: Jogo de damas

```
1  public class TicTacToeClient extends JFrame implements Runnable
2  {
3     private JTextField idField; // campo de texto para a marca
4     private JTextArea displayArea; // JTextArea para a saida
5     private JPanel boardPanel; // painel para o tabuleiro
6     private JPanel panel2; // painel para conter o tabuleiro
7     private Square board[][]; // tabuleiro do jogo-da-velha
8     private Square currentSquare; // quadrado atual
9     private Socket connection; // conexao com o servidor
10    private Scanner input; // entrada a partir do servidor
11    private Formatter output; // saida para o servidor
12    private String ticTacToeHost; // nome do host para o servidor
13    private String myMark; // marca desse cliente
14    private boolean myTurn; // determina de qual cliente e a vez
15    private final String X_MARK = "X"; // primeiro cliente
16    private final String O_MARK = "O"; // segundo cliente
17
18    // configura a interface com o usuario e o tabuleiro
19    public TicTacToeClient( String host )
20    {
21       ticTacToeHost = host; // configura o nome do servidor
22       displayArea = new JTextArea( 4, 30 );
23       displayArea.setEditable( false );
24       add( new JScrollPane( displayArea ), BorderLayout.SOUTH );
25       boardPanel = new JPanel(); // painel para os quadrados
26       boardPanel.setLayout( new GridLayout( 3, 3, 0, 0 ) );
27       board = new Square[ 3 ][ 3 ]; // cria o tabuleiro
28       // faz um loop pelas linhas no tabuleiro
29       for(int row = 0; row < board.length; row++)
30       {  // faz um loop pelas colunas no tabuleiro
31          for(int column=0; column<board[row].length; column++)
32          {  // cria um quadrado
33             board[row][column] = new Square(" ", row*3+column);
34             boardPanel.add( board[ row ][ column ] );
35          }
36       }
37       idField = new JTextField(); // configura o campo de texto
38       idField.setEditable( false );
39       add( idField, BorderLayout.NORTH );
40       panel2 = new JPanel(); // configure o painel
41       panel2.add( boardPanel, BorderLayout.CENTER );
42       add( panel2, BorderLayout.CENTER );
43       setSize( 300, 225 ); // configura o tamanho da janela
44       setVisible( true ); // mostra a janela
45       startClient();
46    }
```

```
47
48    // inicia a thread do cliente
49    public void startClient()
50    {
51       try // conecta-se ao servidor, obtem os fluxos
52       {  // faz uma conexao com o servidor
53          connection = new Socket(
54             InetAddress.getByName(ticTacToeHost), 12345);
55          // obtem os fluxos de entrada e saida
56          input = new Scanner(connection.getInputStream());
57          output= new Formatter(connection.getOutputStream());
58       }
59       catch ( IOException ioException )
60       {
61          ioException.printStackTrace();
62       }
63       // cria e inicia a thread de trabalhador para esse cliente
64       ExecutorService worker = Executors.newFixedThreadPool(1);
65       worker.execute(this); // executa o cliente
66    }
67
68    // thread de controle - atualizacao continua da displayArea
69    public void run()
70    {
71       myMark = input.nextLine(); // marca do jogador (X ou O)
72       SwingUtilities.invokeLater(
73          new Runnable()
74          {
75             public void run()
76             {  // exibe a marca do jogador
77                idField.setText("You are player \""+myMark+"\"");
78             }
79          }
80       );
81       myTurn = (myMark.equals(X_MARK) ); // vez do cliente?
82       // recebe as mensagens enviadas para o cliente
83       while ( true )
84       {
85          if ( input.hasNextLine() )
86             processMessage(input.nextLine());
87       }
88    }
89
90    // processa as mensagens recebidas pelo cliente
91    private void processMessage( String message )
92    {  // ocorreu uma jogada valida
```

```
93          if ( message.equals( "Valid move." ) )
94          {
95             displayMessage( "Valid move, please wait.\n" );
96             setMark( currentSquare, myMark ); // configura a marca
97          }
98          else if ( message.equals( "Invalid move, try again" ) )
99          {
100            displayMessage(message + "\n"); //jogada invalida
101            myTurn = true; // ainda e a vez desse cliente
102         }
103         else if ( message.equals( "Opponent moved" ) )
104         {
105            int location = input.nextInt(); // obtem a posicao
106            input.nextLine(); // pula uma nova linha
107            int row = location / 3; // calcula a linha
108            int column = location % 3; // calcula a coluna
109            setMark(  board[ row ][ column ],
110               ( myMark.equals( X_MARK ) ? O_MARK : X_MARK ) );
111            displayMessage( "Opponent moved. Your turn.\n" );
112            myTurn = true; // agora e a vez desse cliente
113         }
114         else
115            displayMessage( message + "\n" ); // exibe a mensagem
116      }
117
118      // manipula outputArea na thread de despacho de eventos
119      private void displayMessage( final String messageToDisplay )
120      {
121         SwingUtilities.invokeLater(
122            new Runnable()
123            {
124               public void run()
125               {
126                  displayArea.append( messageToDisplay );
127               }
128            }
129         );
130      }
131
132      // metodo utilitario para configurar a marca sobre o ↩
         tabuleiro na thread de despacho de eventos
133      private void setMark(final Square squareToMark, final String ↩
         mark)
134      {
135         SwingUtilities.invokeLater(
136            new Runnable()
```

```
137             {
138                public void run()
139                {
140                   squareToMark.setMark( mark );
141                }
142             }
143          );
144       }
145
146       // envia mensagem para o servidor indicando o quadrado
147       public void sendClickedSquare( int location )
148       {  // se for minha vez
149          if ( myTurn )
150          {
151             output.format( "%d\n", location ); // envia a posicao
152             output.flush();
153             myTurn = false; // nao e minha vez
154          }
155       }
156
157       // configura o Squareatual
158       public void setCurrentSquare( Square square )
159       {
160          currentSquare = square;
161       }
162
163       // classe interna privada para os quadrados no tabuleiro
164       private class Square extends JPanel
165       {
166          private String mark; // marca a ser desenhada
167          private int location; // posicao do quadrado
168          public Square( String squareMark, int squareLocation )
169          {
170             mark = squareMark; // configura a marca
171             location = squareLocation; //posicao desse quadrado
172             addMouseListener(
173                new MouseAdapter()
174                {
175                   public void mouseReleased( MouseEvent e )
176                   {
177                      setCurrentSquare( Square.this );
178                      // envia a posicao desse quadrado
179                      sendClickedSquare( getSquareLocation() );
180                   }
181                }
182             );
```

```
183         }
184
185         // retorno o tamanho preferido de Square
186         public Dimension getPreferredSize()
187         {
188            return new Dimension( 30, 30 );
189         }
190
191         // retorna o tamanho minimo de Square
192         public Dimension getMinimumSize()
193         {
194            return getPreferredSize();
195         }
196
197         // configura a marca para Square
198         public void setMark( String newMark )
199         {
200            mark = newMark; // configura a marca do quadrado
201            repaint(); // repinta o quadrado
202         }
203
204         // retorna a posicao de Square
205         public int getSquareLocation()
206         {
207            return location; // retorna a posicao do quadrado
208         }
209
210         // desenha Square
211         public void paintComponent( Graphics g )
212         {
213            super.paintComponent( g );
214            g.drawRect( 0, 0, 29, 29 ); // desenha o quadrado
215            g.drawString( mark, 11, 20 ); // desenha a marca
216         }
217      }
218   }
```

```
1  public class TicTacToeClientTest
2  {
3     public static void main( String args[] )
4     {
5       TicTacToeClient application; // aplicativo cliente
6       // se nao houver nenhum argumento de linha de comando
7       if ( args.length == 0 )
8           application = new TicTacToeClient( "127.0.0.1" );
9       else
```

```
10          application = new TicTacToeClient( args[ 0 ] );
11       application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 }
```

```
1  public class TicTacToeServer extends JFrame
2  {
3     private String[] board = new String[ 9 ]; // tabuleiro
4     private JTextArea outputArea; // para gerar saida das jogadas
5     private Player[] players; // array de Players
6     private ServerSocket server; // socket de servidor
7     private int currentPlayer; // monitora o jogador atual
8     private final static int PLAYER_X = 0; // o primeiro jogador
9     private final static int PLAYER_O = 1; // o segundo jogador
10    private final static String[] MARKS = { "X", "O" }; // marcas
11    private ExecutorService runGame; // executara os jogadores
12    private Lock gameLock; // sincronizacao do jogo
13    private Condition otherPlayerConnected; // esperar jogador
14    private Condition otherPlayerTurn; // para esperar a jogada
15
16    // configura o servidor de tic-tac-toe e a GUI
17    public TicTacToeServer()
18    {
19       super( "Tic-Tac-Toe Server" );
20       // cria ExecutorService com uma thread para cada jogador
21       runGame = Executors.newFixedThreadPool( 2 );
22       gameLock = new ReentrantLock(); // bloqueio para o jogo
23       // variavel de condicao para os dois jogadores
24       otherPlayerConnected = gameLock.newCondition();
25       // variavel de condicao para a jogada do outro jogador
26       otherPlayerTurn = gameLock.newCondition();
27       for ( int i = 0; i < 9; i++ )
28          board[ i ] = new String( "" ); // cria o tabuleiro
29       players = new Player[ 2 ]; // cria array de jogadores
30       currentPlayer = PLAYER_X; // configura o jogador atual
31       try
32       {
33          server = new ServerSocket( 12345, 2 );
34       }
35       catch ( IOException ioException )
36       {
37          ioException.printStackTrace();
38          System.exit( 1 );
39       }
40       outputArea = new JTextArea(); // cria JTextArea para saida
41       add( outputArea, BorderLayout.CENTER );
```

```
42          outputArea.setText( "Server awaiting connections\n" );
43          setSize( 300, 300 ); // configura o tamanho da janela
44          setVisible( true ); // mostra a janela
45       }
46
47       // espera duas conexoes para que o jogo possa ser jogado
48       public void execute()
49       {  // espera que cada cliente se conecte
50          for ( int I = 0; I < players.length; I++ )
51          {
52             try // espera a conexao, cria Player,
53             {
54                players[ I ] = new Player( server.accept(), I );
55                runGame.execute( players[ I ] );
56             }
57             catch ( IOException ioException )
58             {
59                ioException.printStackTrace();
60                System.exit( 1 );
61             }
62          }
63          gameLock.lock(); // bloqueia o jogo
64          try
65          {
66             players[ PLAYER_X ].setSuspended( false ); // retoma X
67             otherPlayerConnected.signal(); // acorda a thread do X
68          }
69          finally
70          {
71             gameLock.unlock(); // desbloqueia o jogo
72          }
73       }
74
75       // exibe uma mensagem na outputArea
76       private void displayMessage( final String messageToDisplay )
77       {  // exibe uma mensagem a partir da thread de despacho
78          SwingUtilities.invokeLater(
79             new Runnable()
80             {
81                public void run()
82                {
83                   outputArea.append( messageToDisplay );
84                }
85             }
86          );
87       }
```

```
88
89    // determina se a jogada e valida
90    public boolean validateAndMove( int location, int player )
91    {  // enquanto nao for o jogador atual, deve esperar a jogada
92       while ( player != currentPlayer )
93       {
94          gameLock.lock(); // bloqueia o jogo
95          try
96          {
97             otherPlayerTurn.await(); // espera a jogada
98          }
99          catch ( InterruptedException exception )
100         {
101            exception.printStackTrace();
102         }
103         finally
104         {
105            gameLock.unlock(); // desbloqueia o jogo
106         }
107      }
108      // se a posicao nao estiver ocupada, faz a jogada
109      if ( !isOccupied( location ) )
110      {
111         board[location] = MARKS[currentPlayer];
112         currentPlayer = (currentPlayer+1)%2; //troca jogador
113         // deixa que novo jogador saiba que a jogada ocorreu
114         players[ currentPlayer ].otherPlayerMoved( location );
115         gameLock.lock(); // bloqueia o jogo
116         try
117         {
118            otherPlayerTurn.signal(); // sinaliza
119         }
120         finally
121         {
122            gameLock.unlock(); // desbloqueia o jogo
123         }
124         return true; // notifica o jogador jogada valida
125      }
126      else // a jogada nao foi valida
127         return false; // notifica o jogador - jogada invalida
128   }
129
130   // determina se a posicao esta ocupada
131   public boolean isOccupied( int location )
132   {
133      if ( board[ location ].equals( MARKS[ PLAYER_X ] ) ||
```

```
134            board [ location ].equals( MARKS[ PLAYER_O ] ) )
135            return true; // posicao esta ocupada
136         else
137            return false; // posicao nao esta ocupada
138      }
139
140      public boolean isGameOver()
141      {
142         return false; // isso e deixado como um exercicio
143      }
144
145      // classe interna que gerencia cada jogador
146      private class Player implements Runnable
147      {
148         private Socket connection; // conexao com o cliente
149         private Scanner input; // entrada do cliente
150         private Formatter output; // saida para o cliente
151         private int playerNumber; // monitora qual jogador
152         private String mark; // marca para esse jogador
153         private boolean suspended = true; // thread esta suspensa
154
155         // configura a thread Player
156         public Player( Socket socket, int number )
157         {
158            playerNumber=number; //armazena o numero do jogador
159            mark = MARKS[playerNumber]; // especifica a marca
160            connection=socket; //armazena o socket para o cliente
161            try // obtem fluxos a partir de Socket
162            {
163               input=new Scanner(connection.getInputStream());
164               output=new Formatter(connection.getOutputStream());
165            }
166            catch ( IOException ioException )
167            {
168               ioException.printStackTrace();
169               System.exit( 1 );
170            }
171         }
172
173         // envia uma mensagem que o outro jogador fez uma jogada
174         public void otherPlayerMoved( int location )
175         {
176            output.format( "Opponent moved\n" );
177            output.format( "%d\n", location ); // envia a jogada
178            output.flush(); // esvazia a saida
179         }
```

```
180
181         // execucao da thread de controle
182         public void run()
183         {  // envia ao cliente a marca (X ou O)
184            try
185            {
186               displayMessage( "Player " + mark + " connected\n" );
187               output.format( "%s\n", mark ); // envia a marca
188               output.flush(); // esvazia a saida
189               // se for o jogador X, espera o outro jogador
190               if ( playerNumber == PLAYER_X )
191               {
192                  output.format( "%s\n%s", "Player X connected", "↩
                      Waiting for another player\n" );
193                  output.flush(); // esvazia a saida
194                  gameLock.lock(); //bloqueia o jogo
195                  try
196                  {
197                     while( suspended )
198                     {  // espera o jogador O
199                        otherPlayerConnected.await();
200                     }
201                  }
202                  catch ( InterruptedException exception )
203                  {
204                     exception.printStackTrace();
205                  }
206                  finally
207                  {
208                     gameLock.unlock(); // desbloqueia o jogo
209                  }
210                  // envia uma mensagem o outro jogador conectou
211                  output.format("Other player connected. Your move↩
                      .\n");
212                  output.flush(); // esvazia a saida
213               }
214               else
215               {
216                output.format("Player O connected, please wait\n");
217                output.flush(); // esvazia a saida
218               }
219               // enquanto jogo nao terminou
220               while ( !isGameOver() )
221               {
222                  int location = 0; // inicializa a posicao
223
```

```
224                 if ( input.hasNext() )
225                    location = input.nextInt(); // obtem a posicao
226                 // verifica uma jogada valida
227                 if ( validateAndMove( location, playerNumber ) )
228                 {
229                    displayMessage( "\nlocation: " + location );
230                    output.format( "Valid move.\n" ); // notifica
231                    output.flush(); // esvazia a saida
232                 }
233                 else // jogada foi invalida
234                 {
235                    output.format( "Invalid move, try again\n" );
236                    output.flush(); // esvazia a saida
237                 }
238              }
239           }
240           finally
241           {
242              try
243              {
244                 connection.close(); // fecha a conexao
245              }
246              catch ( IOException ioException )
247              {
248                 ioException.printStackTrace();
249                 System.exit( 1 );
250              }
251           }
252        }
253
254        // configura se a thread esta ou nao suspensa
255        public void setSuspended( boolean status )
256        {
257           suspended = status; // configura o valor do suspenso
258        }
259     }
260  }
```

```
1  public class TicTacToeServerTest
2  {
3     public static void main( String args[] )
4     {
5       TicTacToeServer application = new TicTacToeServer();
6       application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7       application.execute();
8     }
```

```
9 }
```

No código acima é apresentado um programa de jogo-da-velha controlado por um servidor com múltiplas threads. Desenvolva um programa de damas modelado com base neste código. Os dois usuários devem fazer movimentos alternados. Seu programa deve mediar os movimentos dos jogadores, determinando de quem é a vez e permitindo apenas movimentos válidos. Os próprios jogadores determinarão quando o jogo acabou.

Exercício 6.8: Jogo de xadrez

Desenvolva um programa de jogo de xadrez modelado de acordo com o programa de damas do exercício 6.7.

Exercício 6.9: Jogo de cartas "vinte e um"

Desenvolva um programa de jogo de cartas do tipo "vinte e um" em que o aplicativo servidor distribui as cartas para cada um dos aplicativos cliente. O servidor deve distribuir as cartas adicionais (de acordo com as regras do jogo) para cada jogador quando solicitado.

Exercício 6.10: Jogo de pôquer

Desenvolva um jogo de cartas de pôquer em que o aplicativo servidor dá as cartas para cada um dos aplicativos cliente. O servidor deve distribuir as cartas adicionais (de acordo com as regras do jogo) para cada jogador quando solicitado.