Exercícios 4.23 ao 4.35

4.23 - Dadas as seguintes declarações de variável:

```
double[] readings;
String[] urls;
TicketMachine[] machines;
```

Escreva atribuições que realizam as seguintes tarefas: (a) Fazer a variável 'readings' referenciar um array que seja capaz de manter 60 valores 'double'; (b) Fazer a variável 'urls' referenciar um array que é capaz de manter 90 objetos String; (c) Fazer a variável 'machines' referenciar um array que seja capaz de manter cinco objetos 'TicketMachine'.
R:

```
double[] readings;
readings = new double[60];

String[] urls;
urls = new String[90];

TicketMachine[] machines;
machines = new TicketMachine[5];
```

4.24 - Quantos objetos String são criados pela seguinte declaração?

```
String labels = new String[20];
```

R:
Cria um array com 20 objetos

4.25 - Verifique o que acontece se a condição do loop for for escrita incorretamente utilizando o operador ‘<=’ em 'printHourlyCounts':

```
for(int hour = 0; hour <= hourCount.length; hour++)
```

R:
Erro, escrevendo '<=' ele ultrapassa o número de elementos do array, que vai de 0 a 23.

4.26 - Reescreva o corpo de 'printHourlyCounts' de modo que o loop for seja substituido por um loop while equivalente. Chame o método reescrito para verificar se ele imprime os mesmos resultados que antes.
R:

```
public void printHourlyCountsWhile()
{
     int hour = 0;
     System.out.println("Hr: Count");
     while(hour < hourCounts.length)
     {
          System.out.println(hour + ": " + hourCounts[hour]);
          hour++;
     }
}
```

4.27 - Reescreva o seguinte método a partir da classe 'Notebook' no projeto notebook2 de modo que utilize um loop for em vez do loop while:

```
/**
* Lista todas as notas no bloco de notas
*/
public void listNotes()
{
     int index = 0;
     while(index < notes.size())
     {
          System.out.println(notes.get(index));
          index++;
     }
}
```

R:

```
public void listNotes()
{
     for(index = 0; index < notes.size(); index++)
     {
          System.out.println(notes.get(index));
     }
}
```

4.28 - Complete o método 'numberOfAccesses', abaixo, para contar o número total de acessos registrados no arquivo de log. Complete-o utilizando um loop for para iterar em 'houtCounts':

```
/**
* Retorna o número de acessos registrados no arquivo
* de log
*/
public int numberOfAccesses()
{
     int total = 0;
     // Adiciona o valor em cada elemento de hourCount
     // ao total.


     return total;
}
```

R:

```
public int numberOfAccesses()
{
     int total = 0;
     for(int i = 0; i < hourCounts.length; i++)
     {
          total += hourCounts[i];
     }

     return total;
}
```

4.29 - Adicione seu método 'numberOfAccesses' à classe 'LogAnalyser' e verifique se ele fornece o resultado correto. Dica: Você pode simplificar na verificação fazendo com que o analisador leia somente os arquivos de log contendo algumas linhas de dados. Dessa maneira, você achará mais fácil determinar se seu método fornecerá ou não a resposta correta. A classe 'LogfileReader' possui um construtor com a seguinte assinatura para ler de um arquivo em particular:

```
/**
* Cria um LogfileReader que fornece dados
* a partir de um arquivo de log em particular.
* @param filename O nome do arquivo de dados de log.
*/
public LogfileReader(String filename)
```

R: Criando um outro método contrutor para 'LogAnalyzer', onde pede um parâmetro, o nome do arquivo, e inicia-se a classe 'LogfileReader' passando esse parâmetro. Esse arquivo serve para testes, pois contem apenas algumas linhas de dados.

4.30 - Adicione um método 'busiestHour' ao 'LogfileAnalyser' que retorna a hora mais ocupada. Você pode fazer isso verificando o array 'hourCount' para encontrar o elemento com a maior contagem. Dica: É necessário verificar cada elemento para saber se você localizou a hora mais ocupada? Se for, utilize um loop for.
R:

```
public int busiestHour()
{
     int busy = 0;
     int acessos = 0;
     for(int hora = 0; hora < hourCounts.length; hora++)
     {
          if(busy == 0)
          {
               busy = hora;
               acessos = hourCounts[hora];
          }
          else
          {
               if(hourCounts[hora] >= acessos)
               {
                    acessos = hourCounts[hora];
                    busy = hora;
               }
          }
     }
     return busy;
}
```

4.31 - Adicione um método 'quietestHour' ao 'LogAnalyser' que retorna o número da hora menos ocupada. Nota: Este exercício é quase idêntico ao anterior, mas há uma pequena armadilha para os descuidados aqui. Certifique-se de verificar seu método com alguns dados em que cada hora poussui uma contagem diferente de zero.
R:

```
public int quietesttHour()
{
     int quiet = 0;
     int acessos = 0;
     for(int hora = 0; hora < hourCounts.length; hora++)
     {
          if(quiet == 0)
          {
               quiet = hora;
               acessos = hourCounts[hora];
          }
          else
          {
               if(hourCounts[hora] <= acessos)
               {
                    acessos = hourCounts[hora];
                    quiet = hora;
               }
          }
     }
     return quiet;
}
```

4.32 - Que hora é retornada por seu método 'busiestHour' se mais de uma hora tiver a maior contagem?
R: A última mais ocupada.

4.33 - Adicione um método ao 'LogAnalyser' que localiza qual período de duas horas é o mais ocupado. Retorne o valor da primeira hora desse período.
R:

```
public int doubleBusiestHour()
{
     int busy = 0;
     int acessos = 0;
     for(int hora = 0; hora < hourCounts.length - 1; hora++)
     {
          if(busy == 0)
          {
               busy = hora;
               acessos = hourCounts[hora] +
          hourCounts[hora+1];
          }
          else
          {
               if(hourCounts[hora] + hourCounts[hora+1] >=
          acessos)
               {
                    acessos = hourCounts[hora] +
               hourCounts[hora+1];
                    busy = hora;
               }
          }
     }
     return busy;
}
```

4.34 - Exercício de desafio. Salve o projeto weblog-analyser sob um nome diferente, de modo que você possa desenvolver uma nova versão que realize uma análise mais extensa dos dados disponíveis. Por exemplo, seria útil saber quais dias tendem ser mais calmos que outros - existe algum padrão cíclico de sete dias, por exemplo? A fim de realizar análise de dados diária, mensal ou anualmente, você precisará fazer algunas alterações na classe 'LogEntry'. Ela já armazena todos os valores de uma única linha de log, mas somente os valores de hora e de minuto estão disponíveis via método de acesso. Adicione mais métodos que disponibilizam os campos restantes disponíveis de maneira semelhante. Em seguida adicione um outro conjunto de métodos adicionais de analise para o analisador.
R:

4.35 - Exercício de desafio. Se tiver concluído o exercício anterior, você poderá estender o formado de arquivo de log com campos numéricos adicionais. Por exemplo, os servidores normalmente armazenam um código numérico que indica se um acesso foi bem sucedido ou não: o valor 200 representa um acesso bem-sucedido, 403 significa que o acesso para o documento foi proibido e 404 significa que o documento não pôde ser localizado. Faça com que o analisador forneça as informações sobre a quantidade de acessos bem-sucedidos e malsucedidos. É provável que este exercício seja muito desafiador, pois ele requererá que você faça alterações em cada classe no projeto.
R: