Exercícios 4.11 ao 4.22

4.11 - Altere seu bloco de notas de modo que as notas sejam numeradas iniciando em 1, em vez de 0. Lembre-se de que o objeto 'ArrayList' ainda estará utilizando índices que iniciam de zero, mas você pode apresentar as notas numeradas a partir de 1 em sua listagem. Certifique-se de modificar também 'showNote' e 'removeNote' de maneira correspondente.
R:

```
public void showNote(int noteNumber)
{
    if(noteNumber < 1)
    {
        System.out.println("Esse número não é válido, pois é
menor que um.");
    }
    else if(noteNumber <= numberOfNotes())
    {
        System.out.println(notes.get(noteNumber - 1));
    }
    else
    {
        System.out.println("Esse número não é valido, pois é
maior que o número de notas.");
    }
}

public void removeNote(int numeroNota)
{
    if(numeroNota < 1)
    {
        System.out.println("Esse número não é válido, pois é
menor que zero.");
    }
    else if(numeroNota < numberOfNotes() - 1)
    {
        notes.remove(numeroNota);
    }
    else
    {
    System.out.println("Esse número não é valido, pois é
maior que o número de notas.");
    }
}
```

```
public void listAllNotes()
{
    int i = 0;
    int size = notes.size();
    while (i < size)
    {
        System.out.println((i +1) + ": " + notes.get(i));
        i ++;
    }
}
```

4.12 - Descubra mais um exemplo de conversão de tipo no método 'getLot' da classe 'Auction'.

R:

```
Lot selectedLot = (Lot) lots.get(number-1);
```

4.13 - O que acontece se você tentar compilar a classe 'Auction' sem uma das conversões de tipo? Por exemplo, edite o método 'showLots' de modo que a primeira instrução no corpo do loop while seja:

```
Lot lot = it.next();
```

R: Mensagem de erro ao compilar: "incompatible type - found java.lang.Object but expected Lot"

4.14 - Adicione um método 'close' à classe 'Auction'. Esse método deve iterar pela coleção de lotes e imprimir detalhes de todos os lotes. Para lotes que forem vendidos, os detalhes devem incluir o nome do arrematador vencedor e o valor do lance ganhador. Para lotes que não foram vendidos, imprima uma mensagem que indica esse fato.
R:

```
public void close()
{
    Iterator it = lots.iterator();
    while (it.hasNext())
    {
        Lot lot = (Lot) it.next();
        Bid bid = lot.getHighestBid();
        if (bid != null)
        {
            System.out.println("pessoa: " + bid.getBidder() +
" valor " + bid.getValue());
        }
        else
        {
            System.out.println("Esse lote ainda não foi
vendido!");
        }
    }
}
```

4.15 - Adicione um método 'getUnsold' à classe 'Auction' com a seguinte assinatura
public ArrayList getUnsold()
Esse método deve iterar pelo campo 'lots', armazenando lotes não vendidos em uma nova variável local 'ArrayList'. No final do método, retorne a lista de lotes não vendidos.
R:

```
public ArrayList getUnsold()
{
    ArrayList lista;
    lista = new ArrayList();
    Iterator it = lots.iterator();

    while (it.hasNext())
    {
        Lot lot = (Lot) it.next();
        if(lot.getHighestBid() != null)
        {
            lista.add(lot);
        }
    }
    return lista;
}
```

4.16 - Suponha que a classe 'Auction' inclui um método que possibilita a remoção de um lote do leilão. Assumindo que os lotes restantes não tem seus campos 'lotNumber' alterados quando um lote é removido, que impacto a capacidade de remover lotes teria sobre o método 'getLot'?
R: Poderia resultar em uma numeração "quebrada", com números de lotes faltando.

4.17 - Reescreva 'getLot' de modo que ele não conte com um lote de um número particular sendo armazenado no índice (number - 1) da coleção. Você pode assumir que os lotes sempre são armazenados na ordem de crescimento de seu número de lote.
R: Sem resposta...

4.18 - Adicione um método 'removeLot' à classe 'Auction', com a seguinte assinatura

```
/**
* Remover o lote com o número de lote dado.
* @param number O número do lote a ser removido.
* @return O lote com o número dado ou null se
* não houver esse lote.
*/
public Lot removeLot(int number)
```

R:

```
/**
* Remover o lote com o número de lote dado.
* @param number O número do lote a ser removido.
* @return O lote com o número dado ou null se
* não houver esse lote.
*/
public void removeLot(int number)
{
    if((number >= 1) && (number < nextLotNumber))
    {
        Lot selectedLot = (Lot) lots.get(number-1);
        if(selectedLot.getNumber() != number)
        {
            System.out.println("Erro interno: " + "Número de
lote errado. " + "Número: " + number);
        }
        lots.remove(number);
    }
    else
    {
        System.out.println("Número do lote: " + number + "
não existe.");
    }
}
```

4.19 - Explore o projeto weblog-analizer criando um objeto 'LogAnalyzer' e chamando seu método 'analyzeHourlyData'. Siga isso com uma chamada para seu método 'printHourlyCounts', que imprimirá os resultados da análize. Quais são as horas mais ocupadas do dia?
R:
12h e 22h possuem 8 acessos.
3h, 8h, 14h, 16h e 21h possuem 6 acessos

4.20 - Escreva uma declaração para uma variável de array 'people' que poderia ser utilizada para referenciar um array de objetos 'Person'.
R:
```
Person people[] = new Person[tamanho];
```

4.21 - Escreva uma declaração para uma variável de array 'vacant' que poderia ser utilizada para referenciar um array de valores 'boolean'.
R:
```
boolean vacant[] = new boolean[tamanho];
```

4.22 - Leia a classe 'LogAnalyzer' e identifique todos os lugares onde a variável 'hourCounts' é utilizada. Nessa etapa, não se preocupe com o que todos os usos significam, pois serão explicados nas seções a seguir. Observe a freqüência com que um par de colchetes é utilizado com a variável.
R:

```
hourCounts = new int[24];
hourCounts[hour]++;
for(int hour = 0; hour < hourCounts.length; hour++) {
System.out.println(hour + ": " + hourCounts[hour]);
```