

## 6 – Caso de estudo – O cartão fidelidade



Cartão de fidelização de clientes das distribuidoras de combustível.



Definição em JAVA da classe `CartaoFidelidade`, que deverá apresentar uma funcionalidade semelhante ao referido cartão.

Cada cartão deve possuir um titular e um número de pontos ( $\geq 0$ ) de bonificação.

Cada instância da dita classe deverá ser capaz de responder adequadamente a um conjunto de mensagens correspondentes às operações de:

- crédito de pontos aquando de um abastecimento ou compras
- débito de pontos (troca de pontos por prémios e brindes, se tal for possível)
- consulta do total de pontos do cartão
- consulta do número total de abastecimentos/compras realizados
- consulta do número total de brindes descontados
- apresentação de todos os dados do cartão (cadeia de caracteres)

Comecemos pela definição das variáveis de instância e do (s) método(s)

construtor(es):

```
public class CartaoFidelidade {  
    //VARIÁVEIS DE INSTÂNCIA:  
    private String titular; //nome do possuidor do cartão  
  
    private int pontos, nDebitos, nCreditos; //saldo e  
                                                //número de cada tipo de operação  
  
    // CONSTRUTORES:  
        //reparem no nome (o mesmo da classe) e na  
        //ausência de um valor de retorno  
    //Construtor por omissão. O que acontece?  
    public CartaoFidelidade() {}  
  
    public CartaoFidelidade (String tit) {  
        titular = tit;  
        pontos = 0;  
        nDebitos = 0;  
        nCreditos = 0;  
    }  
  
    public CartaoFidelidade (String tit, int pts) {  
        titular = tit;  
        pontos = pts;  
        nDebitos = 0;  
        nCreditos = 0; //Porquê zero e não um? Depende da aplicação...  
    }  
}
```

Concentremo-nos agora nos métodos que cada instância da classe CartaoFidelidade terá de ser capaz de proporcionar:

[...]

```
//MÉTODOS DE INSTÂNCIA
```

```
//consulta do total de pontos do cartão
```

```
public int getPontos() {  
    return pontos;  
}
```

```
//crédito de pontos aquando de um abastecimento ou compras
```

```
public void creditarPontos(int pontos) {  
    this.pontos += pontos;  
    nCreditos += 1;  
}
```

```
//débito de pontos (troca de pontos por prémios e brindes)
```

```
public void debitarPontos(int pontos) {  
    //PRE: pontos <= this.pontos  
    this.pontos -= pontos;  
    nDebitos += 1;  
}
```

//consulta do número total de abastecimentos/compras realizados

```
public int getNumCompras() {  
    return nCreditos;  
}
```

//consulta do número total de brindes descontados

```
public int getNumTrocas() {  
    return nDebitos;  
}
```

//apresentação de todos os dados do cartão (cadeia de caracteres)

```
public String toString() {  
  
    return "Cliente: " + titular + "\n" +  
        "Saldo do cartão: " + pontos + "\n" +  
        "Número de compras: " + nCreditos +  
        "\t\tNúmero de brindes: " + nDebitos + "\n";  
}  
}
```

Crie e implemente um programa para testar a classe agora definida.

## Apenas um exemplo:

```
public class TesteCartao {
    public static void main(String[] args) {

        final int ptsCarro = 40, ptsBoneca = 80;

        CartaoFidelidade cardA, cardB, cardC;

        int ptsBrindesPorAtribuir, nBrindesDistrib;
        int numCompras;

        cardA = new CartaoFidelidade("Adalberto U.M. Fulano");
        cardB = new CartaoFidelidade("Sicrano E. Beltrano", 50);
        cardC = new CartaoFidelidade();

        cardA.creditarPontos(34);
        cardA.creditarPontos(44);
        cardB.creditarPontos(12);
        cardB.creditarPontos(45);

        if (cardA.getPontos() >= ptsBoneca)
            cardA.debitarPontos(ptsBoneca);
        else
            System.out.println("O seu saldo não lhe permite
realizar esta operação");

        cardB.creditarPontos(44);
```

Programação Orientada a Objectos - 2010/2011; P. Prata, P. Fazendeiro

```
cardB.creditarPontos(34);
cardB.creditarPontos(44);
if (cardB.getPontos() >= 3 * ptsCarro){
    cardB.debitarPontos(ptsCarro);
    cardB.debitarPontos(ptsCarro);
    cardB.debitarPontos(ptsCarro);
    //Porquê assim???
}else
    System.out.println("O seu saldo não lhe permite
realizar esta operação!\n");

    ptsBrindesPorAtribuir = cardA.getPontos() +
cardB.getPontos() + cardC.getPontos();

    nBrindesDistrib = cardA.getNumTrocas() +
cardB.getNumTrocas() + cardC.getNumTrocas();

    numCompras = cardA.getNumCompras() +
cardB.getNumCompras() + cardC.getNumCompras();

    System.out.println("É necessário provisão para " +
ptsBrindesPorAtribuir + " pontos.");
    System.out.println("Já foram distribuídos " +
nBrindesDistrib + " brindes.");
    System.out.println("Os nossos clientes já
efectuaram " + numCompras + " compras.\n");

    System.out.println(cardA.toString());
```

Programação Orientada a Objectos - 2010/2011; P. Prata, P. Fazendeiro

```
System.out.println(cardB.toString());  
System.out.println(cardC.toString());  
}  
}
```

## Resultado da execução:

```
O seu saldo não lhe permite realizar esta operação!  
  
É necessário provisão para 187 pontos.  
Já foram distribuídos 3 brindes.  
Os nossos clientes já efectuaram 7 operações de compra.  
  
Cliente: Adalberto U.M. Fulano  
Saldo do cartão: 78  
Número de compras: 2          Número de brindes: 0  
  
Cliente: Sicrano E. Beltrano  
Saldo do cartão: 109  
Número de compras: 5          Número de brindes: 3  
  
Cliente: null  
Saldo do cartão: 0  
Número de compras: 0          Número de brindes: 0
```



Responda se souber:

- Como obter o valor médio do número de abastecimentos realizados?
- Como obter o valor médio do número de brindes escolhidos?
- Como obter o número de cartões emitidos?
- Como numerar os cartões emitidos?

[... //classe CartaoFidelidade: alguns métodos adicionais]

```
public boolean equals(Object umObjecto) {  
  
    if (umObjecto != null && umObjecto instanceof CartaoFidelidade) {  
  
        return  
  
        titular.equals((CartaoFidelidade) umObjecto).titular) &&  
  
        this.pontos == ((CartaoFidelidade) umObjecto).pontos &&  
  
        this.nCreditos == ((CartaoFidelidade) umObjecto).nCreditos &&  
  
        this.nDebitos == ((CartaoFidelidade) umObjecto).nDebitos;  
  
    }  
  
    else  
  
        return false;  
  
}
```

```
public Object clone() {  
  
    CartaoFidelidade cloneCartao =  
  
        new CartaoFidelidade(this.titular, this.pontos);  
  
    cloneCartao.nCreditos = this.nCreditos;  
  
    cloneCartao.nDebitos = this.nDebitos;  
  
    return cloneCartao;  
  
}
```



## A classe Vector

A principal limitação dos quadros (arrays) advém do seu carácter estático. É necessário estabelecer a dimensão do array aquando da sua definição e não é possível exceder este limite máximo.

Que acontece em problemas (pensem em alguns) para os quais não é possível determinar, à partida, esta dimensão?

O ideal seria utilizar uma estrutura (dinâmica) cuja dimensão se adapte às necessidades de armazenamento durante a execução do programa...

Temos pois duas alternativas: i) implementar uma classe com a funcionalidade pretendida ou ii) (re)utilizar uma classe com as características desejadas, se a mesma já existir!

Neste caso podemos (devemos) optar pela segunda escolha uma vez que no pacote java.util temos disponível a implementação da classe Vector que se distingue dos arrays pelas seguintes características:

- Um vector pode crescer ou decrescer de tamanho.
- Os vectores armazenam objectos (não podem armazenar tipos simples! A menos que sejam “embrulhados” em objectos... Lembrem-se das classes `Integer`, `Double`,...?).
- Um vector pode conter objectos de diferentes tipos.

Em conclusão, a classe vector implementa uma abstracção de dados que representa uma estrutura linear indexada a partir do índice 0 (deste ponto de vista, análoga ao array) sem limite de dimensão.

Programação Orientada a Objectos - 2010/2011; P. Prata, P. Fazendeiro

Vejamos alguns dos métodos da classe Vector (para uma referência completa estudar a API da classe):

```
Vector() // construtor vector vazio, dimensão inicial zero.

Vector(int capacidadeInicial)
// construtor vector vazio, com dimensão inicial.

void addElement(Object elemento)
// adiciona o elemento especificado ao final do vector.

void insertElementAt(Object obj, int indice)
// insere o elemento especificado na posição indice.

void removeElementAt(int indice)
// remove o elemento na posição indice.

void setElementAt(Object obj, int indice)
// substitui o elemento da posição indice pelo objecto dado.

Object elementAt(int indice)
// devolve o componente presente no indice.

void clear() // remove todos os elementos do vector.

Object clone() // devolve uma cópia do vector.

boolean contains(Object elemento)
// verifica se o objecto especificado é um componente deste vector
```

```
Object firstElement()
```

```
// devolve o primeiro componente (índice 0) do vector.
```

```
Object lastElement() // devolve o último componente do vector.
```

```
int indexOf(Object elemento)
```

```
// procura o índice da 1ª ocorrência de elemento (utiliza o método equals).
```

```
int indexOf(Object elemento, int indice)
```

```
// inicia a procura anterior na posição indice.
```

```
boolean isEmpty() // verifica se o vector não tem componentes
```

```
int size() // devolve a dimensão actual.
```

```
boolean equals(Object o) //permite a comparação de 2 vectores.
```



Retomamos o problema do cartão de fidelização e suponhamos que precisamos de um programa para emitir cartões para novos clientes, listar os dados de todos os cartões emitidos e ordenar os cartões por nome de titular.



```
import java.util.Vector;
import myinputs.Ler;
class TesteVectorCartao {
    public static void main(String[] args) {
        Vector novosCartoes = new Vector();
        CartaoFidelidade cartao;
        String nome;
        int opcao;
        do{
            System.out.println("\nGasolinha & Gasolinha, Lda.");
            System.out.println("\nMENU DE OPERAÇÕES\n");
            System.out.println("1. Emitir Cartão");
            System.out.println("2. Listar Cartões");
            System.out.println("3. Ordenar cartões por nome
                                de titular");
            System.out.println("0. Sair");
            opcao = Ler.umInteiro();
            switch(opcao) {
                case 1: /**INTRODUZA O SEU CÓDIGO AQUI,
                        // antes de ver a solução nas páginas seguintes!***
                        break;
                case 2:
                        /**INTRODUZA O SEU CÓDIGO AQUI***
                        break;
                case 3:
                        /**INTRODUZA O SEU CÓDIGO AQUI***
                        break;
            }
        }while (opcao!=0)
    }
}
```

Para nos ajudar na ordenação por selecção vamos utilizar o seguinte método auxiliar:

// Dado um Vector, **v**, devolve a posição do objecto que contém o titular alfabeticamente menor existente na secção limitada por **inicio** e **v.size()-1**.

```
private static int procuraMenorAlfab (Vector v, int
inicio){

    int iMenor = inicio;

    CartaoFidelidade c1, c2;

    for(int i=inicio+1; i < v.size(); ++i){

        c1 = (CartaoFidelidade) v.elementAt(i);

        c2 = (CartaoFidelidade) v.elementAt(iMenor);

        if (c1.getTitular().compareTo(c2.getTitular())<0)

            iMenor = i;

    }

    return iMenor;

}
```

Nota: O método **int compareTo (String)** devolve um inteiro

- =0 se a String que recebe a mensagem é igual à String argumento
- <0 se a String que recebe a mensagem é alfabeticamente menor que a String argumento
- >0 se a String que recebe a mensagem é alfabeticamente maior que a String argumento

Uma possível resolução para o problema proposto é então:

```
do{
    [...]
    case 1: //emitir e guardar um novo cartão no vector novosCartoes
        System.out.println("\n\nNome completo titular: ");
        nome = Ler.umaString();
        cartao = new CartaoFidelidade (nome);
        novosCartoes.addElement (cartao);
        break;

    case 2: //listar o conteúdo do vector novosCartoes
        for(int pos=0; pos < novosCartoes.size(); ++pos){
            cartao =
                (CartaoFidelidade) novosCartoes.elementAt (pos);
            System.out.println(cartao.toString());
        }
        break;

    case 3: //ordenar o vector novosCartoes alfabeticamente por nome de
            // titular dos objectos (cartões) guardados
        for(int pos=0; pos < novosCartoes.size(); ++pos){
            //encontrar o menor na secção [pos .. novosCartoes.size()]/
            int posMenor =
                procuraMenorAlfab (novosCartoes, pos) ;//método AUXILIAR
```

Programação Orientada a Objectos - 2010/2011; P. Prata, P. Fazendeiro

```
// trocar o elemento da posição pos com o elemento da posição
//posMenor, colocando assim o menor elemento na posição pos
cartão =(CartaoFidelidade) novosCartoes.elementAt (pos) ;
novosCartoes.setElementAt (
    novosCartoes.elementAt (posMenor) , pos) ;
novosCartoes.setElementAt (cartão , posMenor) ;
} //for
break ;
} // switch
}while (opcao!=0) ;

[...]
```

 Responda se souber:

- Como evitar que sejam guardados cartões “repetidos”?
- Como eliminar o cartão do “António Silva”?
- Como eliminar todos os cartões com nome de titular começado por W?