CURSO: Ciência da Computação	**DATA: ____/____/ 2013**
PERÍODO: 4º.	**PROFESSOR: Andrey**
DISCIPLINA: Técnicas Alternativas de Programação	**AULA: 07**

APRESENTAÇÃO

Nesta aula vamos discutir como implementar um tipo abstrato de dados usando classes em java, além de apresentar e discutir conceitos relacionados com Coleções em java.

DESENVOLVIMENTO

O exemplo desta aula foi tirado do livro “Java como programar” (Deitel; Deitel, 2001). Este exemplo consiste em duas classes. A classe Time1 que implementa o TAD e a classe TimeTest que usa a classe Time1 para mostrar a hora em seu formato hh:mm:ss.

Tipo Abstrato de Dados é uma representação abstrata de um tipo de dados (dados e operação) desvinculando-o de sua implementação. O uso de Tipo Abstrato de Dados (TAD) tem como vantagens:

- Integridade dos dados
- Facilidade de manutenção
- Reutilização

Um Tipo Abstrato de dados pode ser definido como uma dupla (V, O) onde V é o conjunto dos valores e O é o conjunto das operações.

Em um tipo abstrato de dados os valores (V) não podem ser acessados diretamente, mas somente através da interface do TAD, que é o seu conjunto de operações (O).

A aplicação do conceito de tipo abstrato de dados é usualmente feita em duas em duas etapas:

- Aplicação
- Implementação

Neste exemplo o nosso TAD vai representar o tempo no formato tanto no formato 24h ex: (13:35:15) quanto no formato 12h (01:35:15 PM). Para isso ele vai ter que armazenar o conjunto de valores (V):

- horas
- minutos
- segundos

e o conjunto de operações (O) para:

- alterar o valor do tempo
- retornar o tempo em formato universal (24h)
- retornar o tempo em formato padrão (12h)

O conceito de TAD é muito similar ao conceito de objetos. A sua implementação usando objetos apresenta a vantagem de permitir preservar as suas características básicas como restrição de acesso às informações usando somente a interface do TAD (conceito de encapsulamento).

O TAD do exemplo será implementado com a classe Time1. Esta classe tem os atributos:

```
   private int hour;
   private int minute;
   private int second;
```

para implementar o conjunto de valores do TAD. Para manter a restrição de acesso aos valores do TAD, estes atributos são colocados como private. Além disso, a classe Time1 tem as operações:

```
public void setTime( int h, int m, int s )
public String toUniversalString()
public String toString()
```

para implementar o conjunto de operações do TAD.

Classe Time1

```
import java.text.DecimalFormat;  // usada para formatação de números

public class Time1 extends Object {
   private int hour;     // 0 - 23
   private int minute;   // 0 - 59
   private int second;   // 0 - 59

   // o construtor Time1 inicializa cada variável de instância em zero.
   // Assegura que cada objeto Time1 começa em um estado consistente.
   public Time1()
   {
      this.setTime( 0, 0, 0 );
   }

   // Seta um novo valor para o tempo realizando a checagem da informação.
   // Retorna zero se os valores forem inválidos.
   public void setTime( int h, int m, int s )
   {
      this.hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
      this.minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
      this.second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   }

   // Converte para String no formato universal de tempo
   public String toUniversalString()
   {
      DecimalFormat twoDigits = new DecimalFormat( "00" );

      return twoDigits.format( this.hour ) + ":" +
             twoDigits.format( this.minute ) + ":" +
             twoDigits.format( this.second );
   }

   // Converte para String no formato padrão
   public String toString()
   {
      DecimalFormat twoDigits = new DecimalFormat( "00" );

      return ( (this.hour == 12 || this.hour == 0) ? 12 : this.hour % 12 ) +
             ":" + twoDigits.format( this.minute ) +
             ":" + twoDigits.format( this.second ) +
             ( this.hour < 12 ? " AM" : " PM" );
   }
```

```
}
```

Para exemplificar o uso de um objeto da classe Time1, é construída a classe TimeTest que inicializa um objeto da classe Time1, apresenta seus valores iniciais, altera os valores e os mostra nos dois formatos.

```
import javax.swing.JOptionPane;

public class TimeTest {
   public static void main( String args[] )
   {
      Time1 t = new Time1();  // calls Time1 constructor
      String output;

      output = "The initial universal time is: " +
               t.toUniversalString() +
               "\nThe initial standard time is: " +
               t.toString() +
               "\nImplicit toString() call: " + t;

      t.setTime( 13, 27, 6 );
      output += "\n\nUniversal time after setTime is: " +
                t.toUniversalString() +
                "\nStandard time after setTime is: " +
                t.toString();

      t.setTime( 99, 99, 99 );  // all invalid values
      output += "\n\nAfter attempting invalid settings: " +
                "\nUniversal time: " + t.toUniversalString() +
                "\nStandard time: " + t.toString();

      JOptionPane.showMessageDialog( null, output,
         "Testing Class Time1",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

Coleções em Java

Na versão atual de Java, uma coleção é um objeto que pode agrupar outros objetos. Uma coleção tem por objetivo facilitar a inserção, busca e recuperação destes objetos agrupados sistematicamente.

Benefícios do uso de coleções:

- redução do esforço de programação,
- redução dos custos de implementação,
- aumento da eficiência da geração de códigos,
- interoperabilidade de codificação,
- redução do esforço de aprendizagem, entre outros.

Tipos de coleções:

- **Set** Um conjunto que não pode ter elementos duplicados;
- **List** coleção de objetos ordenados seqüencialmente sob controle do usuário da lista;
- **Queue** para a fila existem disciplinas específicas de inserção, acesso e remoção, e não só estas;
- **Map** é uma facilidade de acesso a objetos através de chaves, cada chave corresponde a um objeto; e finalmente
- **Collection** que é a raiz de todas as interfaces acima, a que representa uma coleção de objetos.

As interfaces mais utilizadas em Java são List e Set. Todas as classes e interfaces de coleções, estão no pacote java.util.

A interface `List` é implementada pelas classes:

- `ArrayList`,
- `LinkedList` e
- `Vector`.

Classe Vector

Na maioria das linguagens de programação, os arrays possuem tamanho fixo. Eles não podem crescer ou encolher em resposta a alterações nas necessidades do programador. A classe Vector fornece os recursos de estruturas de dados do tipo array que podem ser redimensionados dinamicamente. A classe Vector implementa a interface List. O tipo dos objetos de cada coleção pode ser definida através de um template <Classe>.

```
import java.util.*;

public class Main
{
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner teclado = new Scanner(System.in);
        Vector <Cliente> todos = new Vector <Cliente> ();

        //Lê os dados de 5 cliente e coloca no Vector
        for(int i=0; i<3; i++)
        {
            Cliente cli = new Cliente();
            cli.setNome(teclado.nextLine());
            cli.setCNPJ(teclado.nextLine());
            todos.add(cli);
        }

        // pega os dados de 5 clientes do vector e imprime
        for(int i=0; i<3; i++)
        {
            Cliente cli = todos.elementAt(i);
            System.out.println(cli.getNome()+"; "+cli.getCNPJ());
        }
    }

}
```

Um Vector é uma coleção de objetos, indexados, que pode ter um número variável de objetos. Este número pode inclusive aumentar ou diminuir conforme a necessidade durante a execução do programa. Os principais métodos da classe Vector são:

void add(int index, Object element)	Insere o element na posição index no Vector.
boolean add(Object o)	Insere o objeto no final do Vector.
void addElement(Object obj)	Insere o objeto no final do Vector aumentando em 1 o seu tamanho.
Object elementAt(int index)	Retorna o componente da posição especificada.
Object get(int index)	Retorna o componente da posição especificada.
int indexOf(Object elem)	Procura a primeira ocorrencia do objeto no Vector.
boolean isEmpty()	Testa se o Vector não tem componentes.
Object remove(int index)	Remove o elemento da posição especificada pelo índice no Vector.
boolean remove(Object o)	Remove a primeira ocorrência do elemento no Vector.
int size()	Retorna o número de elementos do Vector.

Interface Iterator

Para acessar os elemento de um Vector podemos usar a interface Iterator em vez de acessá-los através das suas posições. Um Iterator é muito usado para percorre uma coleção. Veja o seguinte código:

```
Iterator <Cliente> itr = todos.iterator();
while (itr.hasNext())
{
      System.out.println(itr.next().getNome());
      System.out.println(itr.next().getCNPJ());
}
```

Classe ArrayList

Outra possibilidade está no uso da classe `ArrayList`, semelhante a classe `Vector`. Vejamos um exemplo:

```
import java.util.*;

public class UsoArrayList {
  private String nomes[] = {"fulano", "cicrano", "beltrano"};
  private ArrayList <String> lista;

  public UsoArrayList()
  {
    lista = new ArrayList <String>();
    for (int i=0; i<nomes.length; i++)
      lista.add(nomes[i]);
    lista.add("meu_nome");
    lista.add("seu_nome");
  }

  public void imprime()
```

```
  {
      Iterator <String> it;

      it = lista.iterator();
      while(it.hasNext())
          System.out.println(it.next());
  }

  public static void main(String args[])
  {
    new UsoArrayList().imprime();
  }
}
//UsoArrayList
```

Veja que a interface Collection fornece um método que devolve um Iterator, objeto responsável por buscar novos elementos nas coleções. O Iterator é na verdade uma interface.

ATIVIDADE

a) Construa um TAD para um array de 10 elementos implementando as operações de busca, ordenação e exibição dos elementos do array.
b) Construa um TAD para realizar as operações com frações (exibição da fração, soma, subtração, multiplicação e divisão).
c) Crie um programa em java que leia Strings até que seja digitado a palavra fim, coloque-as em um vector e as imprima em ordem alfabética.
d) Faça a mesma coisa do exercício anterior só que usando ArrayList.
e) Crie uma janeja que permita o cadastro de nomes de usuários e sempre que for clicado OK o nome do usuário é colocado em um Vector. Quando a janela for fechada imprima todos os nomes que foram digitados em ordem alfabética.

BIBLIOGRAFIA BÁSICA

DEITEL, H. M. e DEITEL, P. J.*. Java, como Programar.* Ed. Bookman. Porto Alegre. 2001.

AHO, A.V.; HOPCROFT, J.E.; ULLMAN, J.D. Data Structures and Algorithms. Addison-Wesley, Reading, Massachusetts, 1983.