Exemplo de uso de listas e interfaces

Este exemplo mostra como criar e usar uma biblioteca e também é um exemplo de uso de interfaces e listas.

Primeiro criaremos uma biblioteca seguindo os passos abaixo.

1) Crie um novo projeto com o nome ArrayBib (Arquivo/Novo projeto). Mas na primeira janela em vez de selecionar "**Aplicação Java**" como de costume, selecione a opção "**Biblioteca de Classe Java**".

2) Depois de criado o projeto, é preciso adicionar pelo menos um pacote à nossa biblioteca. Neste caso criaremos dois. Na janela de Projetos expanda o projeto e clique em **Pacotes de Código Fonte** com o botão direito. A seguir escolha a opção **Novo/Pacote Java**. Crie dois pacotes chamados "**buffer**" e "**sorter**".

3) Clique com o botão direito no pacote "**buffer**" recém criado e escolha a opção **Novo/Interface java**. Crie uma interface chamada **PrintableItem**. O código da interface segue abaixo:

```
package buffer;

/**
 *
 * @author Perrotti
 *
 * Exemplo de uso de interface.
 */
public interface PrintableItem
{
    /**
     * Imprime uma linha com os dados relevantes sobre o
     */
    void printItem();
}
```

4) Clique novamente com o botão direito no pacote "**buffer**" e use a opção Novo/Classe Java para criar a classe CircularBuffer. Abaixo o código das classe:

```
package buffer;

/**
 *
 * @author Perrotti
 *
 * Implementa um buffer circular de objetos.
 * Este tipo de estrutura é apropriada para aplicações or
 * existe uma fila de objetos que precisam ser processad
 * ordem de chegada. É uma lista FIFO (first in firs out)
 *
 */
public class CircularBuffer {
    protected int count, first, last;
    protected PrintableItem buff[];

    public CircularBuffer(int maxItems) {
        count = 0;
        first = 0;
        last = 0;
        buff = new PrintableItem[maxItems];
    }

    /*
     * Retorna a quantidade atual de objetos na lista
     */
    public int getCount() {
        return count;
    }


    /*
     * Adiciona um objeto no final da fila
     */
    public boolean addLast(PrintableItem obj)
    {
        if(count >= buff.length)  // buffer lotado
            return false;
        buff[last]= obj;

        // incremento circular, volta a zero quando chega
        last= (last+1) % buff.length;

        count++;
        return true;
    }

    /*
     * Remove (e retorna) o primeiro objeto da fila
     */
    public Object getFirst()
    {
        if(count == 0)   // Se o buffer está vazio
            return null;

        Object obj = buff[first];   // Pega o primeiro da
        buff[first]= null;          // Anula a antiga pos
```

```
                                    // para não sobrar ur

        first= (first+1) % buff.length; // Incremento cir
        count--;

        return obj;  // retorna o objeto
    }

    /*
     * imprime todos os objetos na lista
     */
    void printAll()
    {
        for(int ct=0; ct<count; ct++)
        {
            int i= (first+ct) % buff.length;
            buff[i].printItem();
        }
    }

}// classe CircularBuffer
```

5) Para o pacote "**sorter**" crie a interface **SorteableItem**. O código segue abaixo.

```
package sorter;
import buffer.PrintableItem;

/**
 *
 * @author Perrotti
 * Exemplo de uso de interface
 *
 */
public interface SorteableItem extends PrintableItem
{
    /**
     * Estabelece o critério de comparação entre dois obj
     * e compara o item no objeto com o item no parametro
     * Retorna: 0 se o item no objeto é igual ao item no
     *          1 se o item no objeto é maior que o item
     *         -1 se o item no objeto é menor que o item
     *
     * O parametro sortBy informa qual atributo será usad
     */
    int compare(SorteableItem item, int sortBy);
}
```

6) Ainda no pacote **"sorter"** crie a classe para as constantes **SortConst**:

```
package sorter;

/**
 *
 * @author Perrotti
 *
 * Constantes para o pacote "sorter"
 */
public class SortConst
{
    public static final int sortByDefault = 1;
}
```

7) E finalmente crie a classe principal do pacote, **SorteableList**:

```
package sorter;

/*
 * To change this license header, choose License Headers
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Perrotti
 *
 * Mantem uma lista ordenada de objetos.
 *
 * Uso:
 *
 * setSortBy(): informa o campo que será usado para orde
 * autoSortOn(): Liga a autoordenação. A lista é sempre
 *               ordenada, mesmo apos a inserção de novos
 * downwardOn(): Usa ordem descendente.
 *
 */
public class SorteableList {
    protected SorteableItem list[];
    protected int count;
    protected boolean autoSort;
    protected int sortBy;
    protected boolean downward;

    public SorteableList(int maxItens) {
        list = new SorteableItem[maxItens];
        count = 0;
        autoSort= false;
        downward= false;
        sortBy = SortConst.sortByDefault;
    }


    public void autoSortOn()
    {
```

```
        autoSort= true;
        sort();
    }

    public void autoSortOff()
    {
        autoSort= false;
    }

    public int getSortBy() {
        return sortBy;
    }

    public void setSortBy(int sortBy) {
        this.sortBy = sortBy;
        if(autoSort) sort();
    }

    public void downwardOn()
    {
        downward= true;
        if(autoSort) sort();
    }

    public void downwardOff()
    {
        downward= false;
        if(autoSort) sort();
    }

    public boolean addItem(SorteableItem item)
    {
        if(count >= list.length) return false;

        if(autoSort) insertItem(item);
        else list[count]= item;

        count++;
        return true;
    }

    private void insertItem(SorteableItem item)
    {
        int pos=0;
        while(item.compare(list[pos], sortBy)>=0 && pos<c
        for(int i=count; i>pos; i--)
            list[i]= list[i-1];
        list[pos]= item;
    }

    public void sort()
    {
        for(int i=0; i<count-1; i++)
          for(int j=i+1; j<count; j++)
          {
              if((list[i].compare(list[j], sortBy)>0 &&
                 (list[i].compare(list[j], sortBy)<0 && c
              {
```

```
                SorteableItem tmp = list[i];
                list[i]=list[j];
                list[j]= tmp;
            }

        }

    }

    public void printList()
    {
        for(int i=0; i<count; i++)
            list[i].printItem();
    }

}
```

Clique com o botão direito no nome do projeto e escolha a opção "Construir". Neste ponto a biblioteca já está pronta para ser usada. Agora vamos criar um projeto que faça uso dela.
Crie um novo projeto chamado Enterprise, mas desta vez crie como Aplicação Java. Para usar a biblioteca será preciso criar uma biblioteca para este novo projeto. Siga os seguintes passos:

1) Na aba "Projetos", clique neste projeto com o botão direito e selecione "Propriedades".

2) Na janela que aparece selecione a categoria "Bibliotecas" e depois clique no botão "Adicionar Biblioteca".

3) Será mostrada uma lista de bibliotecas globais que já estão instaladas. Clique no botão Criar para criar uma nova biblioteca global. O nome sugerido MyLibrary é perfeitamente adequado para diferenciar das bibliotecas "oficiais" do java.

4) Depois de escolhido o nome, uma nova janela permite configurar a biblioteca. Na aba Classpath, clique em "Adicionar JAR/Pasta" e indique a pasta onde se encontra o arquivo ArrayBib.jar.

```
public class Enterprise {
```

```
    public static void print(String st)
    {
        System.out.println(st);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here

        SorteableList list;

        list= new SorteableList(10);
        list.addItem(new PessoaSort("João", "Pereira", "1
        list.addItem(new PessoaSort("Abel", "Ramos", "199
        list.addItem(new PessoaSort("Silvio", "Amoreira",

        print("Sem ordenação");
        list.printList();
        print("+---------------------------------------

        print("Ordenada pelo nome");
        list.setSortBy(PessoaConst.sortByName);
        list.sort();
        list.printList();
        print("+---------------------------------------

        print("Ordenada pelo sobrenome");
        list.setSortBy(PessoaConst.sortByLastName);
        list.sort();
        list.printList();
        print("+---------------------------------------

        print("Ordenada pelo sobrenome em ordem descender
        list.setSortBy(PessoaConst.sortByLastName);
        list.downwardOn();
        list.sort();
        list.printList();
        list.downwardOff();
        print("+---------------------------------------

        print("Ordenada pelo nome com ordenação automáti
        list.setSortBy(PessoaConst.sortByName);
        list.autoSortOn();
        list.addItem(new PessoaSort("Daniel", "Flores", "
        list.printList();
        print("+---------------------------------------
    }

}
```

```
import java.util.Date;
import sorter.*;



/**
 *
 * @author Perrotti
 *
 * Classe Pessoa
 * Classe base para o exemplo de uso da biblioteca com i
 * Representa uma classe da aplicação. Abaixo uma impleme
 * básica com 3 atributos, construtor e os getters/sette
 * correspondentes.
 * Reescreve o método toString().
 *
 * Obs: todos os métodos foram gerados com a opção "Inse
 */
public class Pessoa
{
    protected String nome, sobrenome;
    protected String dataNasc;

    public Pessoa(String nome, String sobrenome, String
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.dataNasc = dataNasc;
    }

    @Override
    public String toString() {
        return "Pessoa{" + "nome=" + nome + ", sobrenome=
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSobrenome() {
        return sobrenome;
    }

    public void setSobrenome(String sobrenome) {
        this.sobrenome = sobrenome;
    }

    public String getDataNasc() {
        return dataNasc;
    }

    public void setDataNasc(String dataNasc) {
        this.dataNasc = dataNasc;
```

```
    }
}
```

```
import sorter.*;
/**
 *
 * @author Usuario
 */
public class PessoaConst {
    public static final int sortByLastName = SortConst.sc
    public static final int sortByName     = SortConst.sc
    public static final int sortByDate     = SortConst.sc
}
```

```
import sorter.SorteableItem;
import sorter.SortConst.*;
/**
 *
 * @author Perrotti
 */
public class PessoaSort extends Pessoa implements Sorteab

    public PessoaSort(String nome, String sobrenome, Stri
        super(nome, sobrenome, dataNasc);
    }

    public void print(String st)
    {
        System.out.println(st);
    }

    @Override
    public int compare(SorteableItem item, int sortBy)
    {
        if(!(item instanceof Pessoa))
            return 0;

        Pessoa psItem = (Pessoa)item;

        if(sortBy == PessoaConst.sortByLastName)
            return sobrenome.compareTo(psItem.sobrenome);

        if(sortBy == PessoaConst.sortByName)
            return nome.compareTo(psItem.nome);

        if(sortBy == PessoaConst.sortByDate)
            return dataNasc.compareTo(psItem.dataNasc);

        return 0;
```

```
    }

    @Override
    public void printItem() {
        System.out.println(toString());
    }


}
```