AULA:

Coleções de Objetos

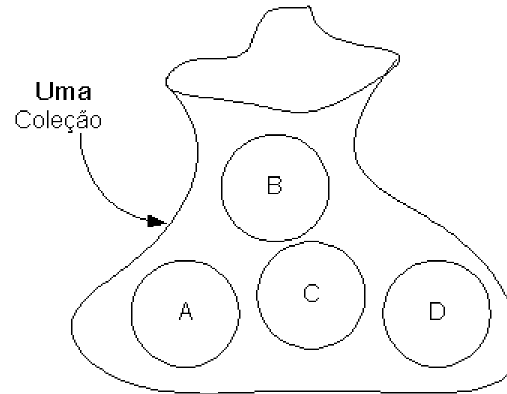
Programação Orientada a Objetos

Alba Lopes, Profa.

http://docentes.ifrn.edu.br/albalopes
alba.lopes@ifrn.edu.br

Coleções de Objetos

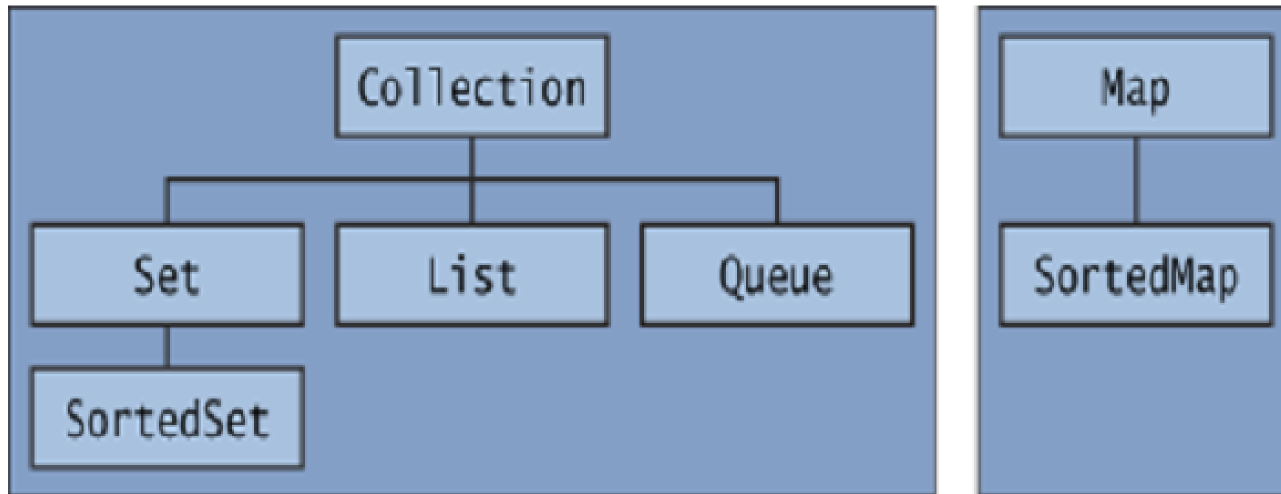
- Um conjunto de objetos
- Sistemas, em geral, precisam manipular vários objetos de uma mesma classe
- Java possui uma API (Application Programming Interface) que fornece interface para coleções



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Coleções de Objetos

- Java fornece vários tipos de coleções:



Tipos de Coleções

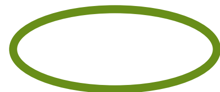
- Conjunto (Set e SortedSet)
  - uma coleção de elementos que não mantém uma ordem nem uma contagem dos elementos
  - Cada elemento ou está no conjunto ou não (não há elementos repetidos)
- Lista (List)
  - uma seqüência de elementos
  - mantém dados a respeito de ambos, a ordem e a contagem.
- Fila (Queue)
  - Fila de elementos
  - Modelo FIFO (First in, first out)
- Mapa (Map e SortedMap)
  - uma associação entre chaves e valores
  - ele mantém um conjunto de chaves e mapeia cada chave para um único valor.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Coleções de Objetos

- A API organiza suas classes da seguinte forma:
  - Uma hierarquia de interfaces
    - Especificações dos vários tipos
  - Uma hierarquia de implementação de classes

Implementations						
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap



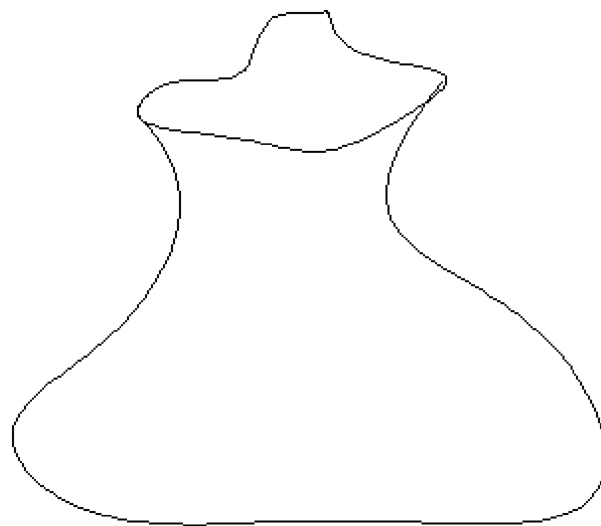
Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Collection

- Métodos comuns para todos os tipos de coleções
- Possui métodos para adicionar, remover e buscar elementos (e mais alguns outros)
  - **size**() – retorna a quantidade de objetos na coleção
  - **isEmpty**() - retorna verdadeiro (true) se a coleção está vazia e falso (false) caso contrário
  - **add** (Objeto o) – adiciona um elemento à coleção
  - **remove** (Objeto o) – remove um objeto da coleção

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Collection



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Collection

moto2

moto1

moto4

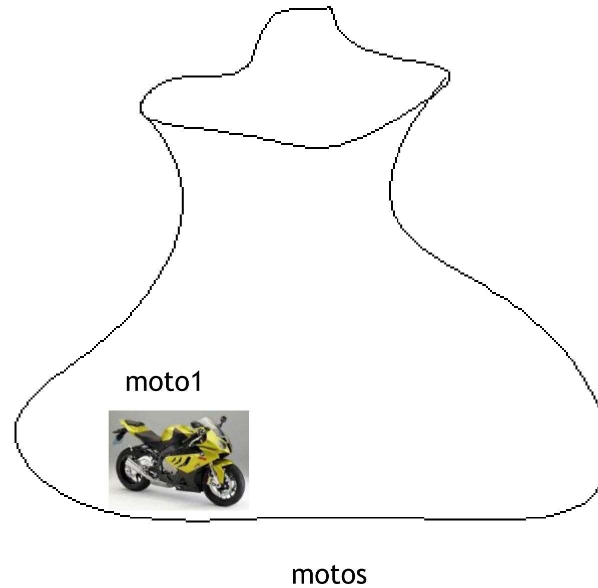
moto5

moto3

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

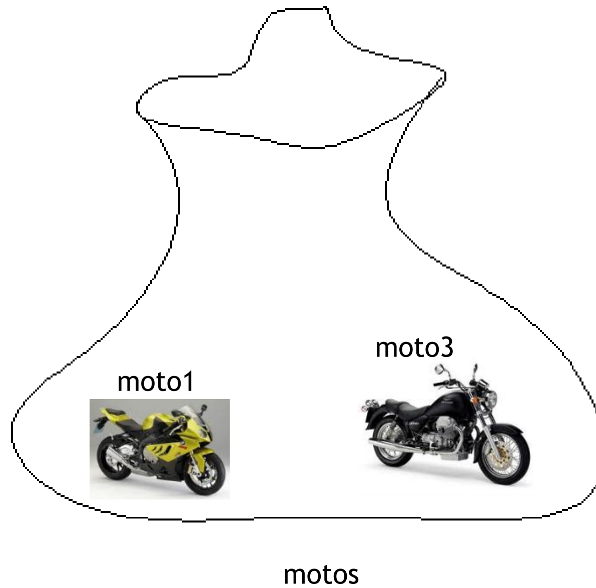
Collection

- motos.**add**(moto1)



Collection

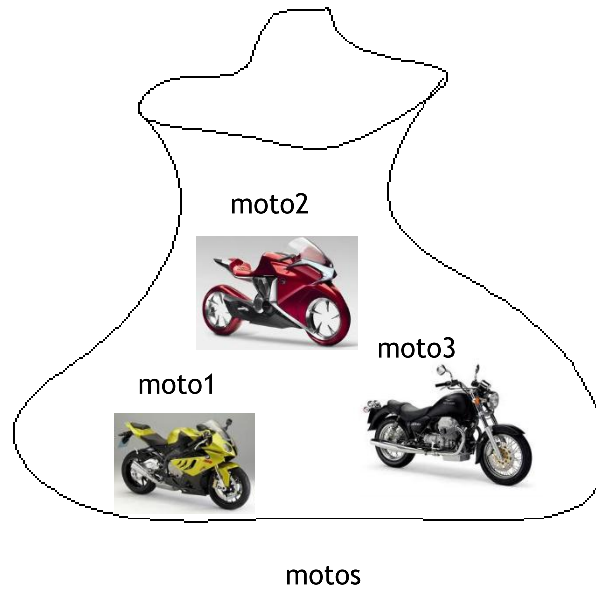
- motos.**add**(moto1)
- motos.**add**(moto3)



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Collection

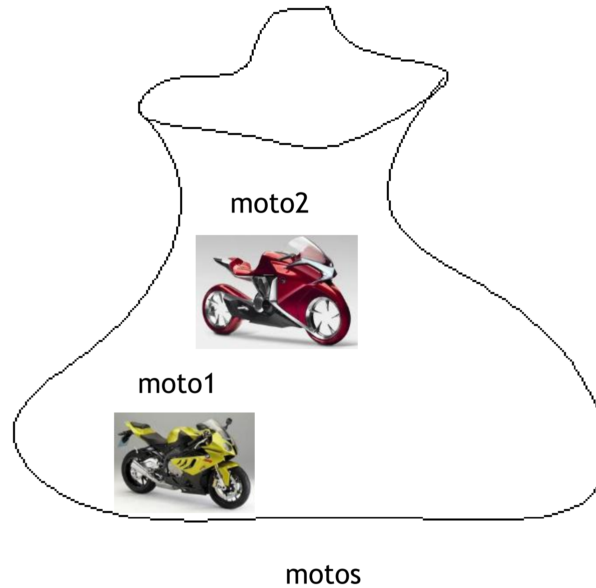
- motos.**add**(moto1)
- motos.**add**(moto3)
- motos.**add**(moto2)



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

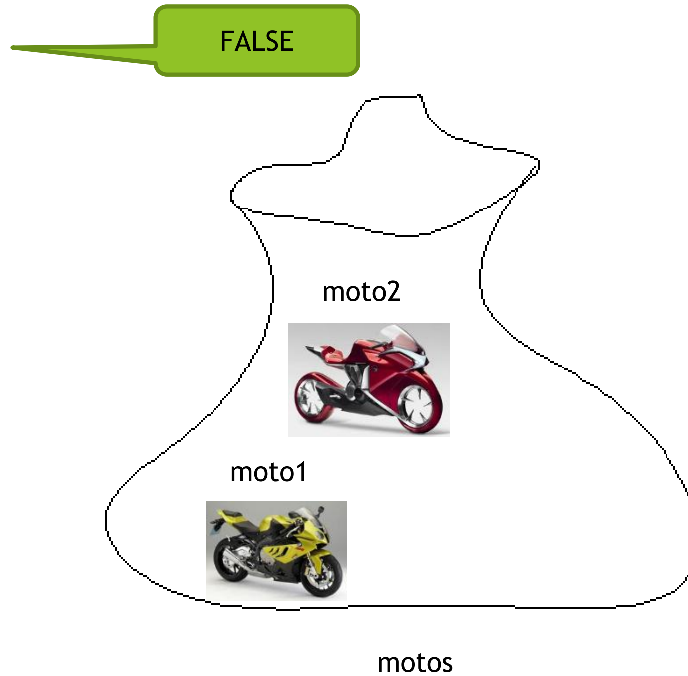
Collection

- motos.**remove**(moto3)



Collection

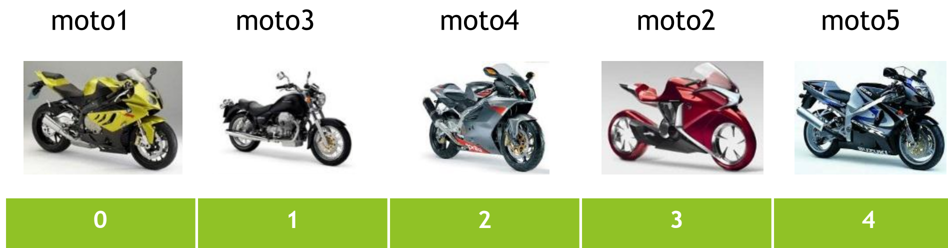
- motos.**isEmpty**()



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- Uma Collection ordenada (também chamada de sequencia
- Pode conter elementos duplicados
- Herda todos os métodos de Collection (size, isEmpty, add, remove, etc) e inclui
- **Representação:**



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- Inclui métodos de acesso posicional
  - **get**(int indice) - retorna elemento na posição indicada pelo parâmetro indice
  - **remove** (int indice) – remove o elemento na posição indicada pelo índice e “fasta” os demais elementos para preencher a posição vazia
  - **add** (Objeto o) – adiciona o objeto depois do último indice
- Os elementos começam no índice 0

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- A lista começa vazia no índice 0

0

motos

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List

- motos.**add**(moto3)

moto1

moto4

moto2

moto5

moto3

0

motos

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List

- motos.**add**(moto3)
- motos.**add**(moto5)

moto1

moto4

moto2

moto3

moto5

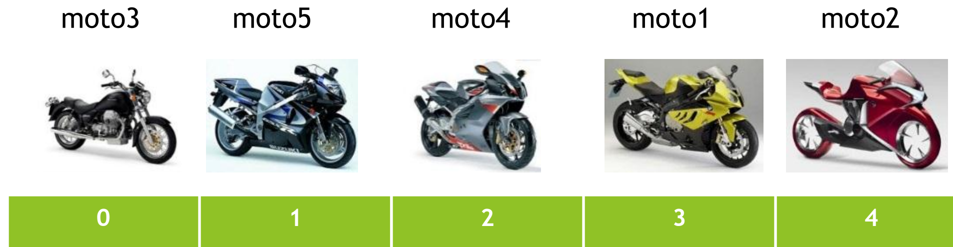
0	1

motos

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- motos.**add**(moto3)
- motos.**add**(moto5)
- motos.**add**(moto4)
- motos.**add**(moto1)
- motos.**add**(moto2)



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- motos.**remove**(0)

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- motos.**size**()

4

moto5	moto4	moto1	moto2
0	1	2	3

motos

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

List (ArrayList)

- Cada elemento tem o seu sucessor (menos o último) e o seu antecessor (menos o primeiro).
- As operações mais importantes em listas são:
  - Adicionar um objeto em qualquer lugar da lista;
  - Remover um objeto de qualquer lugar da lista;
  - Obter o elemento de qualquer lugar da lista;
  - Percorrer os elementos da lista;
  - Verificar se um elemento está na lista;
  - Descobrir o índice de um elemento na lista;
  - Obter o número de elementos da coleção.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo ArrayList

- Crie a classe TestarLista

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

        ArrayList <Motocicleta> motos = new ArrayList<Motocicleta>();

        Motocicleta moto1 = new Motocicleta();
        Motocicleta moto2 = new Motocicleta();

        moto1.setMarca("Honda");
        moto1.setModelo("Titan");

        moto2.setMarca("Yamaha");
        moto2.setModelo("2012");

        motos.add(moto1);
        motos.add(moto2);



    }
}
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- A utilidade de coleções é poder manter um conjunto de dados: adicionando novos valores, buscando valores, removendo valores.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Construindo uma aplicação
  - Vamos construir uma aplicação que seja possível cadastrar motos e verificar as motos cadastradas.
- Altere a aplicação anterior, incluindo novos botões:

Nova Moto
<< Anterior
Próximo >>

Valor dos atributos
Marca:
Modelo:
Velocidade:
Cadastrar

Métodos / Comportamentos
Acelerar
Frear
Está Parada?

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Altere o código fonte da InterfaceMotocicleta para possuir agora uma coleção de objetos do tipo Motocicleta, ao invés de possuir apenas um objeto.
- Inicialize a coleção de objetos no construtor da classe:

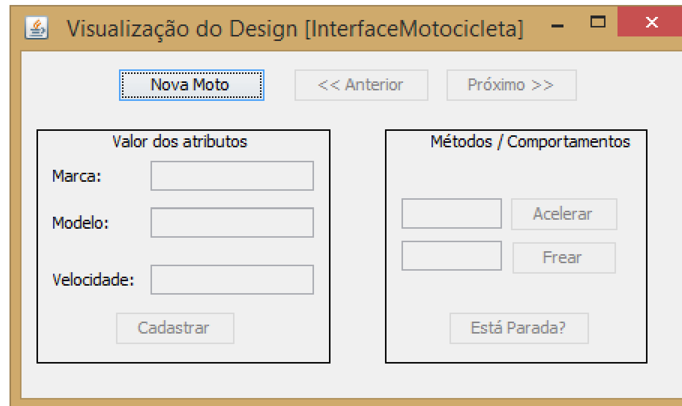
```
public class InterfaceMotocicleta extends javax.swing.JFrame {

    ArrayList<Motocicleta> minhasMotos;
    /**
     * Creates new form InterfaceMotocicleta
     */
    public InterfaceMotocicleta() {
        initComponents();
        minhasMotos = new ArrayList<>();
    }
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

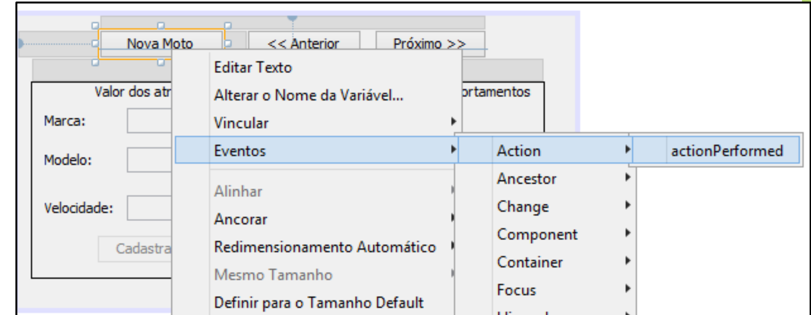
- Modifique a interface para iniciar com todos os campos como desabilitados. Esses campos serão habilitados apenas quando o botão Nova Moto for pressionado:



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Insira o código no botão Nova Moto para habilitar os campos dos atributos e desabilitar os campos dos comportamentos:




```java
private void bNovaMotoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    cModelo.setEnabled(true);
    cMarca.setEnabled(true);
    cVelocidade.setEnabled(true);
    bCadastrar.setEnabled(true);
    bAcelerar.setEnabled(false);
    bFrear.setEnabled(false);
    bParada.setEnabled(false);
    aceleracao.setEnabled(false);
    desaceleracao.setEnabled(false);
}
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Modifique o método actionPerformed do botão Cadastrar para criar um novo objeto do tipo Moto e inseri-lo na coleção:

```
private void bCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    Motocicleta moto = new Motocicleta();
    moto.setMarca(cMarca.getText());
    moto.setModelo(cMarca.getText());
    moto.setVelocidade(Integer.parseInt(cVelocidade.getText()));
    minhasMotos.add(moto);
    cMarca.setEnabled(false);
    cModelo.setEnabled(false);
    cVelocidade.setEnabled(false);
    bCadastrar.setEnabled(false);
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Para poder identificar qual item da coleção está sendo acessado no momento e avançar entre os itens da coleção, crie uma identificação, como por exemplo, mantenha uma referência para o índice do item atual acessado:

```
public class InterfaceMotocicleta extends javax.swing.JFrame {

    private ArrayList<Motocicleta> minhasMotos;
    private int motoAtual;
    /**
     * Creates new form InterfaceMotocicleta
     */
    public InterfaceMotocicleta() {
        initComponents();
        minhasMotos = new ArrayList<>();
        motoAtual = -1;
    }
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Insira uma linha nos demais métodos para recuperar o item atual da coleção:

```
private void bAcelerarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Motocicleta moto = minhasMotos.get(motoAtual);
    moto.acelerar(Integer.parseInt(aceleracao.getText()));
    cVelocidade.setText(String.valueOf(moto.getVelocidade()));
}
```

```
private void bFrearActionPerformed(java.awt.event.ActionEvent evt) {
    Motocicleta moto = minhasMotos.get(motoAtual);
    moto.frear(Integer.parseInt(desaceleracao.getText()));
    cVelocidade.setText(String.valueOf(moto.getVelocidade()));
}
```

```
private void bParadaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Motocicleta moto = minhasMotos.get(motoAtual);
    if (moto.estaParada()){
        JOptionPane.showMessageDialog(this, "A moto está parada!");
    }else{
        JOptionPane.showMessageDialog(this, "A moto não está parada!");
    }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Crie um método para habilitar os botões Próximo e Anterior de forma que seja possível navegar entre os itens da coleção.
  Esse método verifica o valor da variavel motoAtual e compara com o tamanho da coleção de forma a poder habilitar ou desabilita os botões Proximo e Anterior

```
private void habilitarProximoAnterior(){

    if ( (minhasMotos.size() > 0)){
        if (motoAtual > 0 ){
            bAnterior.setEnabled(true);
        }else{
            bAnterior.setEnabled(false);
        }
        if ((motoAtual+1) < minhasMotos.size()){
            bProximo.setEnabled(true);
        }else{
            bProximo.setEnabled(false);
        }
    }
}
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Chame o método de habilitarProximoAnterior no botão Cadastrar:

```
private void bCadastrarActionPerformed(java.awt.event.ActionEvent evt)

    Motocicleta moto = new Motocicleta();
    moto.setMarca(cMarca.getText());
    moto.setModelo(cMarca.getText());
    moto.setVelocidade(Integer.parseInt(cVelocidade.getText()));
    minhasMotos.add(moto);
    cMarca.setEnabled(false);
    cModelo.setEnabled(false);
    cVelocidade.setEnabled(false);
    bCadastrar.setEnabled(false);
    bAcelerar.setEnabled(true);
    bFrear.setEnabled(true);
    bParada.setEnabled(true);
    aceleracao.setEnabled(true);
    desaceleracao.setEnabled(true);
    motoAtual = minhasMotos.size()-1;
    habilitarProximoAnterior();

}
```


Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Inclua ações nos botões Proximo e Anterior para recuperar os itens da coleção, bem como chame o método habilitarProximoAnterior para poder navegar nos itens da coleção:

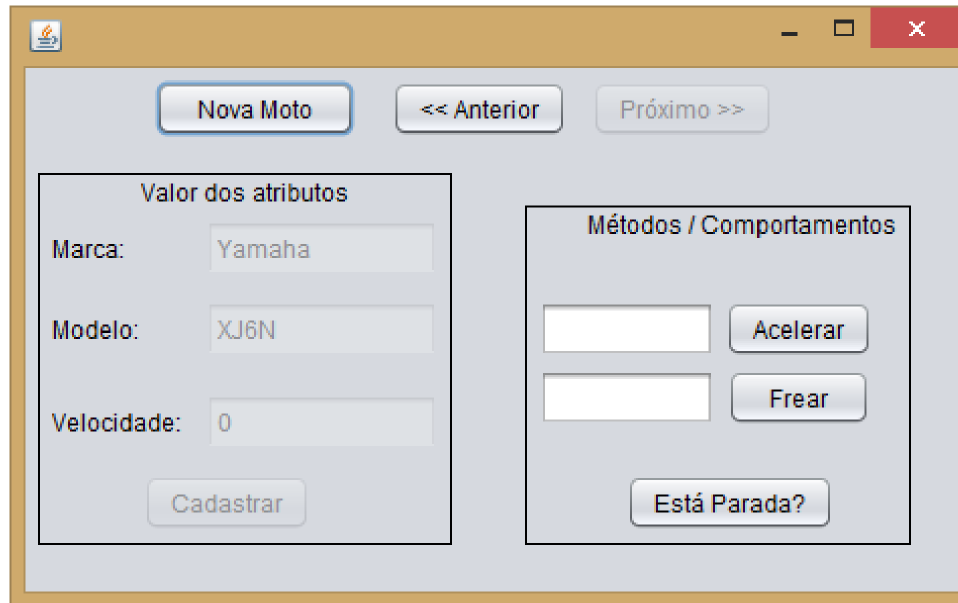
```
private void bAnteriorActionPerformed(java.awt.event.ActionEvent evt) {
    motoAtual-=1;
    Motocicleta moto = minhasMotos.get(motoAtual);
    cMarca.setText(moto.getMarca());
    cModelo.setText(moto.getModelo());
    cVelocidade.setText(String.valueOf(moto.getVelocidade()));
    habilitarProximoAnterior();
}

private void bProximoActionPerformed(java.awt.event.ActionEvent evt) {
    motoAtual+=1;
    Motocicleta moto = minhasMotos.get(motoAtual);
    cMarca.setText(moto.getMarca());
    cModelo.setText(moto.getModelo());
    cVelocidade.setText(String.valueOf(moto.getVelocidade()));
    habilitarProximoAnterior();
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Utilizando Coleções

- Teste a aplicação criada!



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

- Crie uma coleção de objetos do tipo ContaCorrente e altere a aplicação criada anteriormente de forma que seja possível navegar entre os itens da coleção:

Visualização do Design [IContaCorrente]

Nova Conta | << Anterior | Próximo >>

Dados do cliente:

Nome:

Saldo Inicial:

Limite:

Cadastrar

Operações

Saldo: R$

Operação: Sacar Depositar

Valor:

Realizar Operação

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br