

Exemplo de Herança, Polimorfismo, Interfaces e *ArrayList*.

1. Considere a classe *Cão* com um campo de dados *nome*, construtor que recebe o *nome* como parâmetro e 2 métodos: *ladra* e *caçaGatos*.

```
public class Cão {  
    private String nome;  
    public Cão(String nome) {  
        this.nome = nome;  
    }  
    public void ladra() {  
        System.out.println(nome + " a ladrar.");  
    }  
    public void caçaGatos() {  
        System.out.println(nome + " a caçar gatos.");  
    }  
}
```

Construa uma classe *Teste* apenas com método *main*, no qual cria 2 cães, Bart e Fido. Coloque-os num *array* de 10 elementos e invoque o método *ladra*:

```
public class Teste {  
    public static void main(String[] args){  
        Cão c1 = new Cão("Bart");  
        Cão c2 = new Cão("Fido");  
        Cão[] cães = new Cão[10];  
        cães[0] = c1;  
        cães[1] = c2;  
        for(int i=0; i<cães.length; i++)  
            if(cães[i] != null)  
                cães[i].ladra();  
    }  
}
```

2. Na classe *Cão* coloque um atributo *tamanho* do tipo *int*.

```
public class Cão {  
    ...  
    private int tamanho;  
    ...  
}
```

3. Modifique o método *ladra* da classe *Cão* para acrescentar:

```
public void ladra() {  
    String som;  
    if (tamanho > 60) som = "Uff! Uff!";  
    else if (tamanho > 20) som = "Ruff! Ruff!";  
    else som = "Ip! Ip!";  
    System.out.println(nome + " a ladrar: " + som);  
}
```

Copie este código, insira na classe *Cão*, marque o código inserido, e com o botão direito do rato selecione: *Format (Alt+Shift+F)*.

4. Comece por executar o método *main* da classe *Teste*. Verifique que o campo de dados *tamanho* é usado, mesmo sem ser inicializado.

Em seguida coloque os tamanhos de Bart e Fido no método *main*.

```
public class Teste {  
    public static void main(String[] args){  
        ...  
        cães[0] = new Cão("Bart");  
        c1.tamanho(70);  
        cães[1] = new Cão("Fido");  
        c2.tamanho(8);  
        ...  
    }  
}
```

Verifique o problema do acesso e a mensagem de erro:

```
"tamanho has private access in Cão"
```

5. Coloque métodos para acesso ao campo de dados *tamanho* (obter e alterar) na classe *Cão* (métodos *getter* e *setter*).

```
public class Cão {  
    ...  
    public int getTamanho() {  
        return tamanho;  
    }  
    public void setTamanho(int tamanho) {  
        this.tamanho = tamanho;  
    }  
    ...  
}
```

Em seguida coloque os tamanhos de Bart e Fido no método main usando o respetivo método *set*.

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        cães[0] = new Cão("Bart");  
        c1.setTamanho(70);  
        cães[1] = new Cão("Fido");  
        c2.setTamanho(8);  
        ...  
    }  
}
```

Execute.

6. Volte a apagar os métodos de acesso *getter* e *setter* e use o *Refactor* do NetBeans.

Com o botão direito do rato sobre o atributo *tamanho*, selecione:

- *Refactor > Encapsulate Fields ...*
- *Refactor*

7. Acrescente o método *ladra(int num)* na classe *Cão* (*overloading*, sobrecarga de métodos):

```
public class Cão {  
    ...  
    public void ladra() { ... }  
    public void ladra(int num) {  
        while( num > 0) {  
            ladra();  
            num = num -1;  
        }  
    }  
    ...  
}
```

No método *main* da classe *Teste*, invoque *ladra(2)*.

```
public class Teste {  
    public static void main(String[] args){  
        ...  
        for(int i=0; i<cães.length; i++)  
            if(cães[i] != null)  
                cães[i].ladra(2);  
    }  
}
```

8. Acrescente mais um construtor sem parâmetros na classe *Cão*.

```
public class Cão {  
    ...  
    public Cão() { }  
    ...  
}
```

9. No método *main* crie um cão através do construtor sem parâmetros, coloque no *array* e execute.

```
public class Teste {  
    public static void main(String[] args){  
        ...  
        Cão c3 = new Cão();  
        ...  
        cães[2] = c3;  
        ...  
    }  
}
```

Verifique que os campos de dados são usados, mesmo sem atribuir valores. Coloque métodos de acesso para o campo de dados *nome*. Dê um nome e tamanho ao cão criado.

```
public class Cão {  
    ...  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    ...  
}
```

```
public class Teste {  
    public static void main(String[] args){  
        ...  
        Cão c3 = new Cão();  
        c3.setNome("Snoopy");  
        c3.setTamanho(30);  
        ...  
    }  
}
```

10. Substitua o ciclo *for* do método *main* pelo ciclo *for* da versão Java 5.0, designado "for each":

```
public class Teste {  
    public static void main(String[] args){  
        ...  
        for(Cão c : cães)  
            if(c != null)  
                c.ladra(2);  
    }  
}
```

Significa: para cada elemento "c" do tipo *Cão* no *array* *cães*, se for diferente de *null*, execute a ação *ladra(2)*.

11. Os *arrays* têm tamanho fixo. Para colocar um objeto num *array*, esse objeto tem de ser atribuído a um índice específico. A remoção ou inserção de um elemento numa determinada posição obriga a deslocamento de outros elementos.

ArrayList é uma classe da biblioteca de classes da linguagem Java, utilizada para servir de contentor de objetos. Os objetos são guardados sob o tipo de dados *Object* e retribuídos também como *Object*. Para aceder às propriedades dos objetos guardados é necessário efetuar o *downcasting* para o tipo de dados correto.

ArrayList possui muitos métodos, sendo os principais:

<code>public boolean add(Object elem)</code>	adiciona o objeto <i>elem</i> ao <i>ArrayList</i>
<code>public int size()</code>	retorna o número de elementos do <i>ArrayList</i>
<code>public Object get(int index)</code>	retorna o objeto guardado no índice indicado como parâmetro (mas sob o tipo de dados <i>Object</i>)
<code>public Object remove(int index)</code>	remove o objeto guardado no índice enviado como parâmetro
<code>public boolean remove(Object elem)</code>	remove o objeto <i>elem</i> se ele existir no <i>ArrayList</i>
<code>public boolean contains(Object elem)</code>	retorna true se o objeto <i>elem</i> existir no <i>ArrayList</i>
<code>public int indexOf(Object elem)</code>	retorna o índice do objeto <i>elem</i> , se existir no <i>ArrayList</i> , ou -1
<code>public boolean isEmpty()</code>	retorna true se o <i>ArrayList</i> não tiver elementos

A classe *ArrayList* está contida no *package java.util*.

Reescreva novamente o método *main* da classe *Teste* usando a classe *ArrayList*.

```
import java.util.ArrayList;

public class Teste {
    public static void main(String[] args){
        Cão c1 = new Cão("Bart");
        c1.setTamanho(70);
        Cão c2 = new Cão("Fido");
        c2.setTamanho(8);

        ArrayList cães = new ArrayList();
        cães.add(c1);
        cães.add(c2);
        for(int i=0; i<cães.size(); i++) {
            Cão c = (Cão) cães.get(i);
            c.ladra(2);
        }
    }
}
```

12. Modifique o ciclo *for* por um ciclo "for each":

```
public class Teste {
    public static void main(String[] args){
        ...
        for(Object o: cães) {
            Cão c = (Cão) o;
            c.ladra(2);
        }
    }
}
```

13. Coloque no *ArrayList* uma *String*.

```
public class Teste {
    public static void main(String[] args){
        ...
        cães.add(c1);
        cães.add(c2);

        cães.add("String é um tipo diferente do tipo Cão");
        ...
    }
}
```

Verifique que dá erro apenas em tempo de execução:

*Exception in thread "main" java.lang.ClassCastException: java.lang.String
cannot be cast to Cão at Teste.main*

14. Use o *ArrayList* do Java 5.0. No Java 5.0 os *ArrayList* são parametrizados.

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        ArrayList<Cão> cães = new ArrayList<Cão>();  
        ...  
    }  
}
```

Verifique que dá erro em tempo de compilação:

```
no suitable method found for add(String)  
  method ArrayList.add(int,Cão) is not applicable  
    (actual and formal argument lists differ in length)  
  method ArrayList.add(Cão) is not applicable  
    (actual argument String cannot be converted to Cão by method invocation conversion)  
-----  
(Alt-Enter shows hints)
```

15. Apague a colocação da *String* no *ArrayList* e altere o código do método *main* da classe *Teste*.

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        ArrayList<Cão> cães = new ArrayList<Cão>();  
  
        cães.add(c1);  
        cães.add(c2);  
  
        for(Cão c: cães) {  
            c.ladra(2);  
        }  
  
        for(int i=0; i<cães.size(); i++) {  
            Cão c = cães.get(i);  
            c.ladra(2);  
        }  
    }  
}
```


16. Suponhamos que precisávamos de representar vários animais entre os quais cães, gatos e leões, com as seguintes características:

Atributos:

- String nome – o nome do animal.

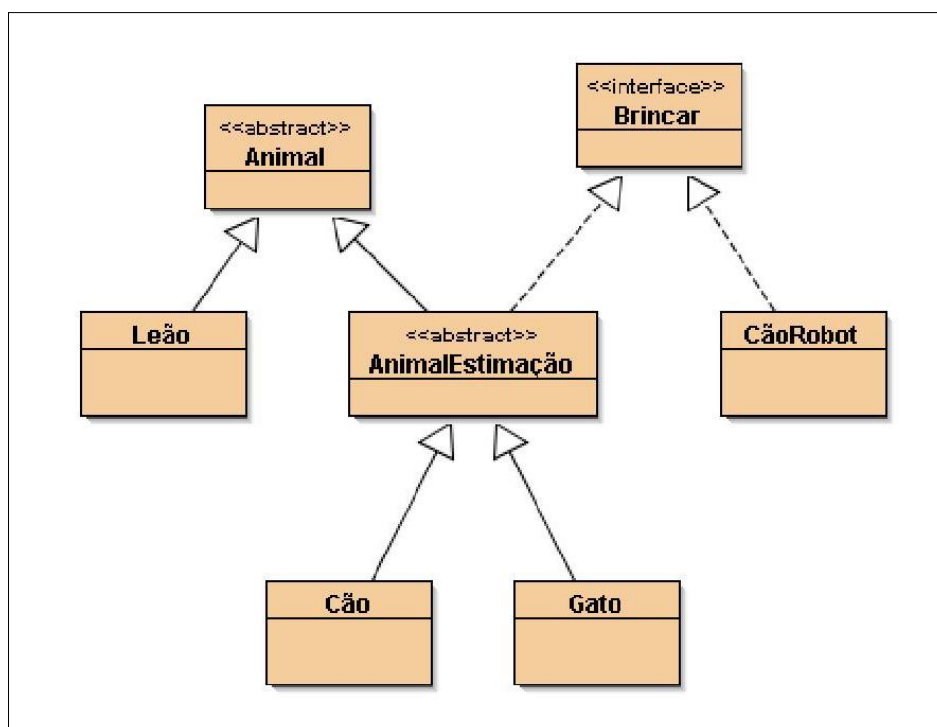
Comportamentos:

- come() - Comportamento quando o animal come.
- fazRuído() - Comportamento quando o animal faz ruído.

Os cães e os gatos por serem animais de estimação ainda apresentam um comportamento comum: *brinca()*.

Um *Cão Robot* também apresenta este comportamento *brinca()* mas não pertence à mesma árvore hierárquica dos cães, gatos e leões pois não come.

Implemente as classes e interfaces seguintes:



```
public abstract class Animal {
    private String nome;
    public Animal(String nome) { this.nome = nome; }
    public String getNome() { return nome; }
    public abstract void fazRuído();
}
```

```
public class Leão extends Animal {
    public Leão(String nome) {
        super(nome);
    }
    public void fazRuído() {
        System.out.println( getNome() + " a rugir... " );
    }
}
```

```
public interface Brincar {
    void brinca();
}
```

```
public abstract class AnimalEstimacão extends Animal
                                   implements Brincar {
    public AnimalEstimacão(String nome) {
        super(nome);
    }
    public void brinca() {
        System.out.println( getNome() + " a brincar." );
    }
}
```

```
public class Cão extends AnimalEstimacão {
    public Cão(String nome) {
        super(nome);
    }
    public void fazRuído() {
        System.out.println( getNome() + " a ladrar." );
    }
    public void caçaGatos() {
        System.out.println( getNome() + " a caçar gatos." );
    }
}
```

```
public class Gato extends AnimalEstimacão {
    public Gato(String nome) { super(nome); }
    public void fazRuído() {
        System.out.println( getNome() + " a miar... " );
    }
    public void caçaRatos() {
        System.out.println( getNome() + " a caçar ratos." );
    }
}
```

```
public class CãoRobot implements Brincar {

    String nome;
    public CãoRobot(String nome) { this.nome = nome; }
    public void brinca() {
        System.out.println( nome + " a brincar." );
    }
}
```

```
import java.util.ArrayList;
public class Teste {
    public static void main(String[] args) {
        Cão c1 = new Cão("Bart"); Cão c2 = new Cão("Fido");
        Gato g1 = new Gato("Bolinhas");
        CãoRobot r1 = new CãoRobot("CãoRobot");
        Leão l1 = new Leão("Simba");

        ArrayList animais = new ArrayList();

        animais.add(c1); animais.add(c2); animais.add(g1);
        animais.add(r1); animais.add(l1);

        for (int i=0; i<animais.size(); i++) {
            Brincar b = (Brincar) animais.get(i);
            b.brinca();
        }
    }
}
```

Não dá erro de compilação.

Dá erro de execução:

Exception in thread "main" java.lang.ClassCastException: Leão cannot be cast to Brincar

Se alterarmos o *ArrayList* para só permitir conter objetos *Brincar*:

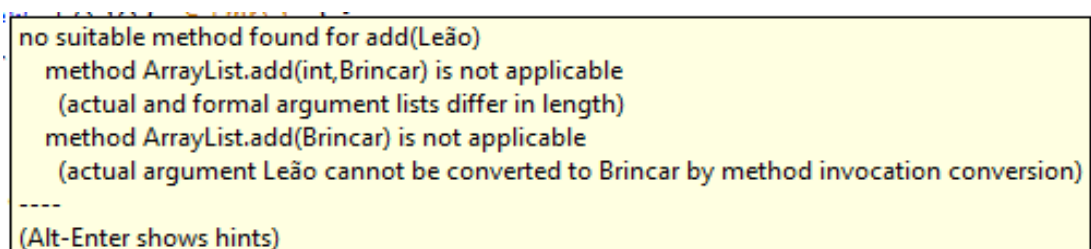
```
import java.util.ArrayList;
public class Teste {
    public static void main(String[] args) {
        Cão c1 = new Cão("Bart");
        Cão c2 = new Cão("Fido");
        Gato g1 = new Gato("Bolinhas");
        CãoRobot r1 = new CãoRobot("CãoRobot");
        Leão l1 = new Leão("Simba");

        ArrayList<Brincar> animais = new ArrayList<Brincar>();

        animais.add(c1);
        animais.add(c2);
        animais.add(g1);
        animais.add(r1);
        animais.add(l1); // erro de compilação

        for (int i=0; i<animais.size(); i++) {
            Brincar b = animais.get(i);
            b.brinca();
        }
    }
}
```

A verificação de compatibilidade de tipos é feita em tempo de compilação e dá erro de compilação.



```
no suitable method found for add(Leão)
method ArrayList.add(int,Brincar) is not applicable
(actual and formal argument lists differ in length)
method ArrayList.add(Brincar) is not applicable
(actual argument Leão cannot be converted to Brincar by method invocation conversion)
-----
(Alt-Enter shows hints)
```