

Exercício Animais

1. Crie um projeto no *Netbeans*, designado *Animais*, onde serão adicionadas versões simplificadas de algumas classes.
2. Adicione a classe *Cao* com o atributo *nome*, construtor que recebe, como parâmetro, um *nome* e dois métodos: *ladra* e *cacaGatos*.

Repare que o uso da instrução *System.out.println* nos métodos, serve apenas para demonstrar a execução desses métodos. No entanto, esta instrução não deve ser aplicada nos métodos de classes que modelam (representam) entidades.

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

3. Adicione uma classe *TesteAnimais* com método *main*, no qual cria 2 cães, *Bart* e *Fido*. Coloque-os num contentor do tipo *array*, de 10 elementos e invoque o método *ladra*.

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

4. Na classe *Cao* adicione o atributo *tamanho* do tipo *int*.

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

5. Analise o seguinte código para o método *ladra*.

```
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 para a classe *Cao*, substituindo o código já existente. Selecione o código copiado, carregue no botão direito do rato e selecione: *Format (Alt+Shift+F)*.

6. Execute o método *main* da classe *TesteAnimais*. Verifique que o atributo *tamanho* é usado, mesmo sem ser inicializado.

Em seguida tente alterar os tamanhos a *Bart* e *Fido*, no método *main*.

```
public class TesteAnimais {  
    public static void main(String[] args){  
        Cao c1 = new Cao("Bart");  
        Cao c2 = new Cao("Fido");  
        c1.tamanho=70;  
        c2.tamanho=8;  
        Cao[] caes = new Cao[10];  
        ...  
    }  
}
```

Verifique a mensagem de erro que descreve o problema:

```
"tamanho has private access in Cao"
```

7. Adicione métodos para manipular (aceder e alterar) o atributo *tamanho*, da classe *Cao* (métodos *getter* e *setter*).

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

Em seguida atribua os tamanhos de *Bart* e *Fido* no método *main*, usando para o efeito, o método *setTamanho*.

```
public class TesteAnimais {  
    public static void main(String[] args){  
        Cao c1 = new Cao("Bart");  
        Cao c2 = new Cao("Fido");  
        c1.setTamanho(70);  
        c2.setTamanho(8);  
        Cao[] caes = new Cao[10];  
        ...  
    }  
}
```

Execute.

8. Apague os métodos *getTamanho* e *setTamanho*, para de seguida utilizar o *Refactor* do *NetBeans*.

Clique no botão direito do rato sobre o atributo *tamanho*, e selecione:

- *Refactor > Encapsulate Fields... (Ctrl+Alt+Shift+E)*
- *Refactor*

9. Adicione o método *ladra(int numeroVezes)* na classe *Cao* (*overloading*, sobrecarga de métodos).

```
public class Cao {  
    ...  
    public void ladra() { ... }  
    public void ladra(int numeroVezes) {  
        while(numeroVezes > 0) {  
            ladra();  
            numeroVezes--;  
        }  
    }  
    ...  
}
```

No método *main* da classe *TesteAnimais*, substitua *ladra()* por *ladra(2)*.

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

10. Adicione um construtor sem parâmetros na classe *Cao*.

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

11. No método *main*, da classe *TesteAnimais*, crie um cão através do construtor sem parâmetros e armazene-o no *array*.

```
public class TesteAnimais {  
    public static void main(String[] args){  
        ...  
        Cao c3 = new Cao();  
        ...  
        caes[2] = c3;  
        ...  
    }  
}
```

Execute e verifique que os atributos são inicializados, mesmo sem a atribuição explícita de valores.

Adicione métodos *getter* e *setter* para o atributo *nome*, na classe *Cao*.

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

Atribua um *nome* e *tamanho* ao cão criado.

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

12. Substitua o ciclo *for* do método *main* pelo ciclo *for* designado “*for each*”.

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

Significa: para cada elemento “*c*” do tipo *Cao* do *array* *caes*, se “*c*” for diferente de *null*, executa a ação *ladra(2)*.

13. Os *arrays* têm tamanho fixo. Para armazenar um objeto num *array*, esse objeto tem de ser colocado numa posição específica. A remoção ou inserção, num *array*, de um elemento numa determinada posição obriga ao deslocamento de outros elementos.

ArrayList<E> é uma classe nativa do Java, utilizada para servir de contentor de objetos. O parâmetro *E* permite definir o tipo de objetos do *ArrayList*. Os objetos são armazenados como sendo do tipo de dados *E* e obtidos também como sendo do tipo de dados *E*.

ArrayList<E> possui um conjunto de métodos, dos quais se destacam:

Assinatura	Descrição
public boolean add (E e)	Adiciona ao final do <i>ArrayList</i> , o objeto <i>e</i> , recebido por parâmetro.
public int size ()	Retorna o número de elementos do <i>ArrayList</i> .
public E get (int <i>index</i>)	Retorna o objeto (sob o tipo de dados <i>E</i>) existente na posição <i>index</i> , recebida como parâmetro, do <i>ArrayList</i> .
public E remove (int <i>index</i>)	Elimina o objeto existente na posição <i>index</i> , recebida como parâmetro, do <i>ArrayList</i> . Retorna, sob o tipo de dados <i>E</i> , o objeto eliminado.
public boolean remove (Object <i>o</i>)	Elimina a primeira ocorrência do objeto <i>o</i> , recebido por parâmetro, se ele existir no <i>ArrayList</i> . Retorna <i>true</i> , se o objeto foi eliminado.
public boolean contains (Object <i>o</i>)	Retorna <i>true</i> se o objeto <i>o</i> , recebido por parâmetro, existir no <i>ArrayList</i> .
public int indexOf (Object <i>o</i>)	Retorna a posição da primeira ocorrência do objeto <i>o</i> , recebido por parâmetro, se existir no <i>ArrayList</i> . Se não existir retorna -1.
public boolean isEmpty ()	Retorna <i>true</i> se o <i>ArrayList</i> estiver vazio.

A classe *ArrayList<E>* pertence ao package *java.util*.

14. Altere, novamente, o método *main* da classe *TesteAnimais*, utilizando agora um contentor do tipo *ArrayList*.

```
import java.util.List;
import java.util.ArrayList;

public class TesteAnimais {

    public static void main(String[] args){

        Cao c1 = new Cao("Bart");
        Cao c2 = new Cao("Fido");
        Cao c3 = new Cao();
        c1.setTamanho(70);
        c2.setTamanho(8);
        c3.setNome("Snoopy");
        c3.setTamanho(30);

        List<Cao> caes = new ArrayList<>();

        caes.add(c1);
        caes.add(c2);
        caes.add(c3);

        for(int i=0; i<caes.size(); i++) {

            Cao c = caes.get(i);

            c.ladra(2);

        }

    }

}
```

15. Substitua o ciclo *for* por um ciclo “for each”.

```
public class TesteAnimais {

    public static void main(String[] args){

        ...

        for(Cao c: caes) {

            c.ladra(2);

        }

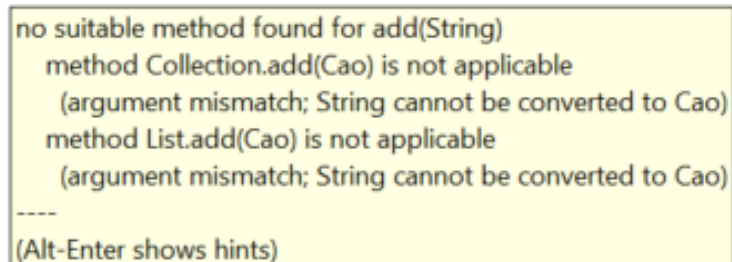
    }

}
```


16. Tente adicionar ao *ArrayList* uma *String*.

```
public class TesteAnimais {  
    public static void main(String[] args){  
        ...  
        caes.add(c1);  
        caes.add(c2);  
        caes.add(c3);  
  
        caes.add("String é um tipo diferente do tipo Cao");  
        ...  
    }  
}
```

Verifique que ocorre o seguinte erro de compilação:



no suitable method found for add(String)
method Collection.add(Cao) is not applicable
(argument mismatch; String cannot be converted to Cao)
method List.add(Cao) is not applicable
(argument mismatch; String cannot be converted to Cao)

(Alt-Enter shows hints)

Elimine, do código, a adição da *String* ao *ArrayList*.

17. Suponhamos que precisamos de representar os seguintes animais: cães, gatos e leões. Todos eles têm em comum:

Atributo:

- *nome* – o nome do animal.

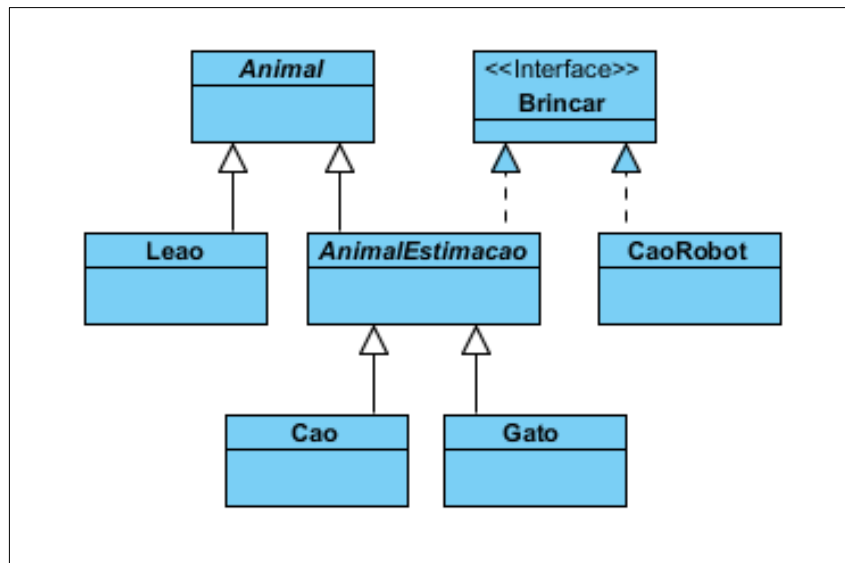
Método (comportamento):

- *fazRuido()* - comportamento quando o animal faz ruído.

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

Para além destes animais, também precisamos de representar o *CaoRobot*. O *CaoRobot*, tal como os animais de estimação, possui o comportamento *brinca()*. No entanto, não pertence à mesma árvore hierárquica dos animais pois não faz ruído.

Implemente as seguintes classes e interface:



```
public abstract class Animal {
    private String nome;

    public Animal(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public abstract void fazRuido();
}
```

```
public class Leao extends Animal {
    public Leao(String nome) {
        super(nome);
    }

    public void fazRuido() {
        System.out.println(getNome() + " a rugir... ");
    }
}
```

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

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

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

```
public class CaoRobot implements Brincar {  
    private String nome;  
    public CaoRobot(String nome) {  
        this.nome = nome;  
    }  
    public void brinca() {  
        System.out.println(nome + " a brincar.");  
    }  
}
```

Na classe *Cao* faça as alterações destacadas a negrito e outras que considere relevantes.

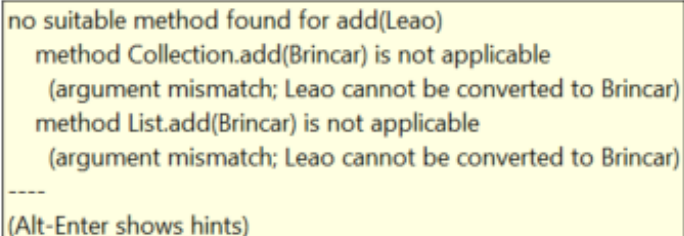
```
public class Cao extends AnimalEstimacao {  
    ...  
    public Cao(String nome) {  
        super(nome);  
    }  
    public Cao() {  
        super("sem nome");  
    }  
    ...  
    public void fazRuido() {  
        System.out.println(getNome() + " a ladrar.");  
    }  
}
```

18. Adicione uma nova classe, com o método *main*, designada *TesteBrincar*.
19. Nesta classe deve criar dois objetos do tipo *Cao*, *Bart* e *Fido*, um objeto do tipo gato, *Bolinhas*, um objeto do tipo *CaoRobot*, *AIBO*, e um objeto do tipo *Leao*, *Simba*.
Armazene num contentor, do tipo *ArrayList*, todos os objetos criados que têm o comportamento *brinca()*. Na inicialização do *ArrayList*, tem que indicar que o tipo de objetos deste contentor é (a interface) *Brincar*.
Tente adicionar ao contentor todos os objetos criados, pela seguinte ordem: *Bart*, *Fido*, *Bolinhas*, *AIBO* e *Simba*.

```
import java.util.List;
import java.util.ArrayList;
public class TesteBrincar {
    public static void main(String[] args) {
        Cao c1 = new Cao("Bart");
        Cao c2 = new Cao("Fido");
        Gato g1 = new Gato("Bolinhas");
        CaoRobot r1 = new CaoRobot("AIBO");
        Leao l1 = new Leao("Simba");

        List<Brincar> animais = new ArrayList<>();
        animais.add(c1);
        animais.add(c2);
        animais.add(g1);
        animais.add(r1);
        animais.add(l1); // erro de compilação
        for (Brincar b: animais) {
            b.brinca();
        }
    }
}
```

A verificação de compatibilidade de tipos, no *ArrayList*, é feita em tempo de compilação e, ocorre um erro de compilação ao tentar adicionar ao contendor um objeto que não é do tipo *Brincar*.



```
no suitable method found for add(Leao)
method Collection.add(Brincar) is not applicable
(argument mismatch; Leao cannot be converted to Brincar)
method List.add(Brincar) is not applicable
(argument mismatch; Leao cannot be converted to Brincar)
----
(Alt-Enter shows hints)
```

Assim, são evitados erros de execução no ciclo “for each”.

20. Remova as instruções relacionadas com o leão e execute o método *main*.