Polimorfismo

- Polimorfismo é a capacidade de se referenciar a um objeto por meio de formas diferentes.
  - Um objeto tem somente uma forma.
  - Uma variável de referência pode se referir a objetos de diferentes formas.

```
Animal a = new Gato();
```

- Quando fazemos isso, estamos olhando para o objeto Gato como um Animal.
  - Dessa forma, poderemos chamar todos os métodos de Animal, mas nenhum de Gato. Pode parecer pouco realista. Mas há motivos para se querer fazer isso.
- O Java, assim como a maioria das linguagens OO, permite que você se refira a um objeto com uma variável que seja um dos tipos da classe pai.

```
Object a = new Gato();
```

Herança e arrays

- Bom motivo para usar o polimorfismo

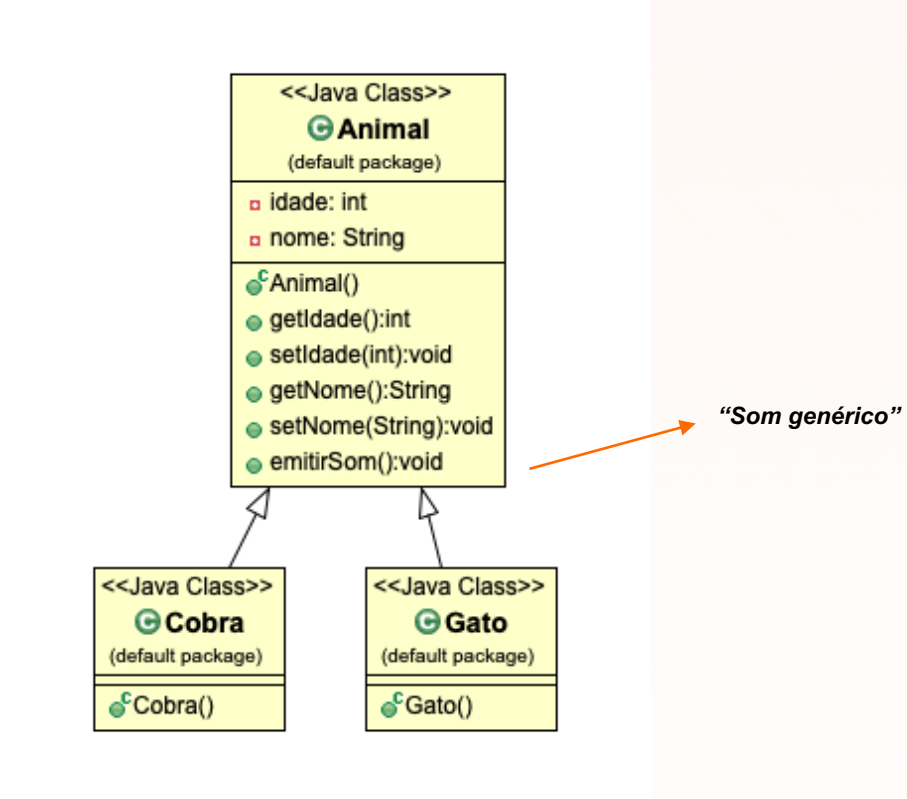
```java
public class Gato {
    private String nome;
    private String cor;
    private int idade;

    public Gato() {
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getCor() {
        return cor;
    }
    public void setCor(String cor) {
        this.cor = cor;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public void miar() {
        System.out.println("Gato miando...");
    }
}
```

```java
public class Cobra {
    private String nome;
    private String cor;
    private int idade;

    public Cobra() {
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getCor() {
        return cor;
    }
    public void setCor(String cor) {
        this.cor = cor;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public void sibilar() {
        System.out.println("Cobra sibilando...");
    }
}
```

```java
public class HerancaArrays {

    public static void main(String[] args) {

        Gato [] corrida1 = {new Gato(), new Gato(), new Gato()};
        Cobra [] corrida2 = {new Cobra(), new Cobra()};

        for (int i = 0; i < corrida1.length; i++) {
            corrida1[i].miar();
            corrida2[i].sibilar(); //erro
        }
    }
}
```

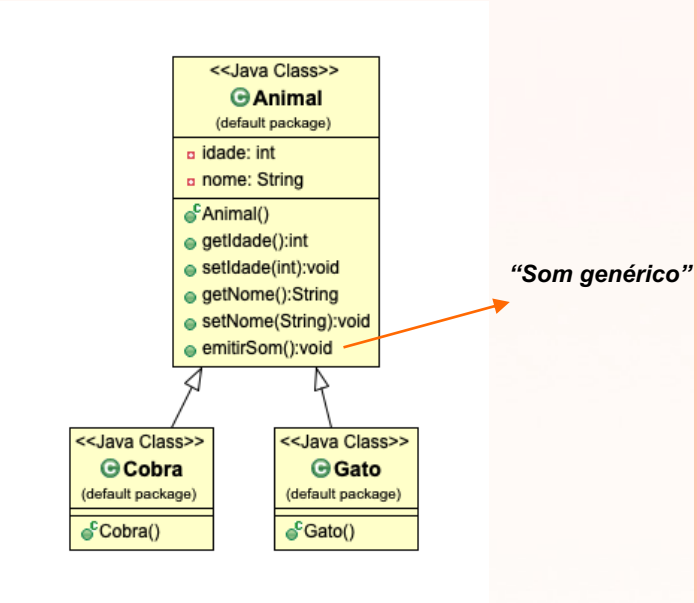
Herança e arrays

- Bom motivo para usar o polimorfismo



Herança e arrays

- Bom motivo para usar o polimorfismo




```
1
2  public class HerancaArrays {
3
4      public static void main(String[] args) {
5          Animal [] corrida = {new Gato(), new Cobra(), new Cobra(), new Gato(), new Gato()};
6          for (int i = 0; i < corrida.length; i++) {
7              corrida[i].emitirSom();
8          }
9      }
10 }
```

```
Problems  Javadoc  Declaration  Console
<terminated> HerancaArrays [Java Application] /Users/gustavo/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2.v20200815-0932/jre/bin/java
Som genérico
Som genérico
Som genérico
Som genérico
Som genérico
```

Sobrescrita de método

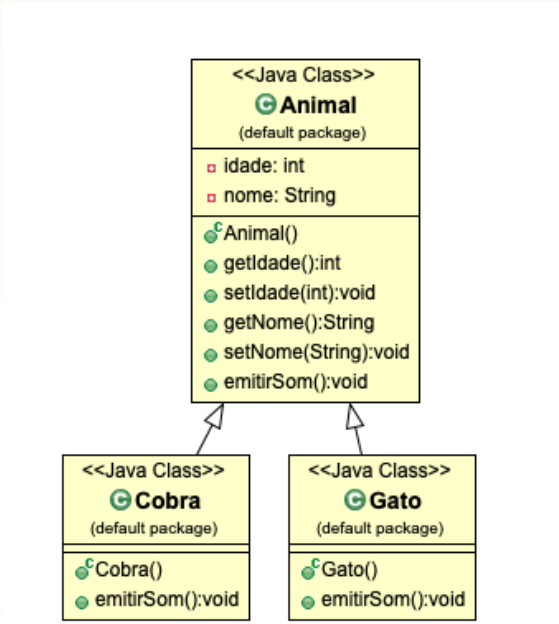
- Além de produzir uma nova classe com base em uma antiga incluindo recursos adicionais, você pode modificar o comportamento atual da classe filha.

- Se um método é definido em uma subclasse, de modo que o nome, tipo de retorno e lista de argumentos de um método da classe filha seja igual ao nome, tipo de retorno e lista de argumentos de um método da classe pai, diz-se que o novo método sobrepôs ou sobrescreveu o antigo.

Sobrescrita de método

- Por exemplo:

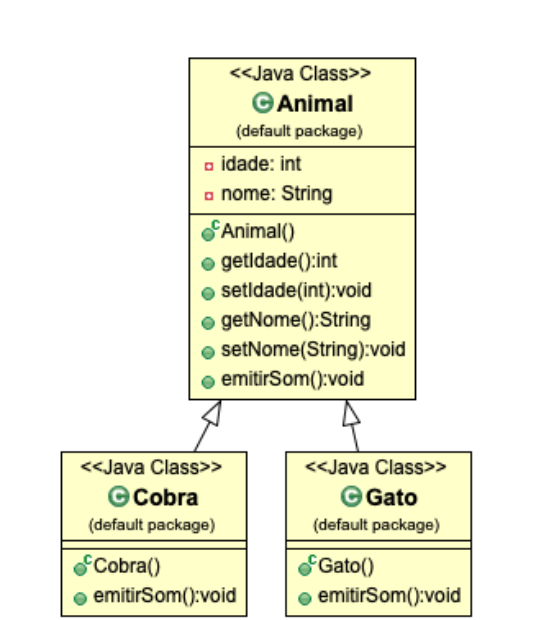
```java
public class Animal{
    private int idade;
    private String nome;
    public Animal() {
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void emitirSom() {
        System.out.println("Som genérico");
    }
}
```




```java
public class Cobra extends Animal{
    public void emitirSom() {
        System.out.println("sssssss");
    }
}
```

```java
public class Gato extends Animal{
    public void emitirSom() {
        System.out.println("Miauuu");
    }
}
```

Sobrescrita de método




```java

public class HerancaArrays {

    public static void main(String[] args) {
        Animal [] corrida = {new Gato(), new Cobra(), new Cobra(), new Gato(), new Gato()};
        for (int i = 0; i < corrida.length; i++) {
            corrida[i].emitirSom();
        }
    }
}
```

```
Problems  Javadoc  Declaration  Console
<terminated> HerancaArrays [Java Application] /Users/gustavo/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2.v20200815-0932/jre/bin/java
Miauuu
ssssss
ssssss
Miauuu
Miauuu
```

POLIMORFISMO - RECAPTULANDO

- O Java, assim como a maioria das linguagens OO, na verdade, permite que você se refira a um objeto com uma variável que seja de um dos tipos da superclasse. Assim sendo, você pode escrever o seguinte:

- Animal m = new Gato();
- m. [control + espaço]

- Usando a variável Animal como referência ao objeto Gato, você só pode acessar as partes do objeto que fazem parte de Animal ou de suas superclasses.
- As partes específicas de Gato estão ocultas. No que diz respeito ao compilador, Gato é um Animal, não um Gato.

Chamada de métodos

- Observe o cenário abaixo:

- Cachorro a = new Cachorro();
- Gato g = new Gato();

- Vamos supor que você tenha sobrescrito o métoto emitirSom() em Gato e em Cachorro. Ao carregar a.emitirSom() e g.emitirSom(), você chama comportamentos diferentes. O objeto Cachorro executa a versão emitirSom() associada à classe Cachorro, e o objeto Gato executa a versão de emitirSom() associada a classe Gato. Menos óbvio é o seguinte:

- Animal e = new Cachorro();
- e.emitirSom();

Chamada de métodos

- Na verdade, você tem o comportamento associado ao objeto ao qual a variável se refere no runtime.
- Esse momento não é determinado pelo tipo da variável no momento da compilação.
- Esse é o aspecto do polimorfismo, e uma característica importante das linguagens OO. Esse comportamento é muitas vezes denominado de chamada de método virtual. No exemplo anterior, o método e.emitirSom() executado é do tipo real do objeto, um Cachorro.

Argumentos polimórficos

- Você pode escrever métodos que aceitam um objeto "genérico" (neste caso, a classe Animal), e pode trabalhar adequadamente com os objetos de qualquer subclasse desse objeto. Você pode produzir um método em uma classe que alimenta um Animal. Usando recursos polimórficos, você pode fazer o seguinte:

```
public static void testarSom(Animal e){
    e.emitirSom();
}
...main... {
  Gato a = new Gato();
  Animal x = new Cachorro();
  Animal k = new Gato();
  testarSom (a);
  testarSom (x);
  testarSom (k);
}
```

O Operador instanceof

- Se você receber um objeto que usa uma referência do tipo Animal, ele pode
- "ser visto" como um Gato ou uma Cobra. Você pode testar usando o operador instanceof da seguinte forma:

```
public void facaAlgo (Animal e) {
    if (e instanceof Cobra) {
        System.out.println("sou uma cobra");
    } else {
        System.out.println("sou um outro tipo de animal");
    }
}
```

Casting de objetos

- Em circunstâncias nas quais você recebeu uma referência para uma classe pai, e usando o operador instanceof, determinou que o objeto na verdade é uma subclasse particular, você pode restaurar a plena funcionalidade do objeto fazendo o casting da referência.

```
public void facaAlgo (Animal e) {
   if (e instanceof Gato) {
      Gato e2 = (Gato) e;
      e2.ronronar(); //método implementado apenas na classe Gato
    }
}
```

- Se você não fizer o casting, uma tentativa de executar e.ronronar() não terá sucesso porque o compilador não pode localizar esse método na classe Animal. Se você não fizer o teste usando o instanceof, o casting pode falhar.

Casting de objetos

- Os casts feitos para cima na hierarquia de classe são sempre permitidos, de fato, não exigem o operador de casting. Podem ser feitos por meio de atribuição simples.

  - Gato g = new Gato();
  - Animal a = g; // Não precisa escrever Animal a = (Animal) g;

- Nos casts "para baixo", o compilador precisa aceitar que o cast é pelo menos possível. Por exemplo, qualquer tentativa de cast de uma referência Gato em uma referência Cobra é definitivamente proibida, porque Gato não é um Cobra. A classe para a qual o cast é feito deve ser alguma subclasse do tipo de referência corrente.

- Se o compilador permitir o cast, o tipo de objeto é verificado em runtime.

Sobrescrevendo métodos de Object

```
1
2  public class HerancaArrays {
3
4      public static void main(String[] args) {
5          Animal [] corrida = {new Gato(), new Cobra(), new Cobra(), new Gato(), new Gato()};
6          corrida[0].setNome("Felix");
7          corrida[1].setNome("C1");
8          corrida[1].setNome("C2");
9          corrida[3].setNome("Tom");
10         corrida[4].setNome("Mingau");
11         for (int i = 0; i < corrida.length; i++) {
12             Animal a = corrida[i];
13             System.out.println(a);
14             System.out.println(a.toString());
15             a.emitirSom();
16         }
17     }
18 }
19
```

```
Gato@1963006a
Gato@1963006a
Miauuu
Cobra@6d9c638
Cobra@6d9c638
ssssss
Cobra@7dc5e7b4
Cobra@7dc5e7b4
ssssss
Gato@1ee0005
Gato@1ee0005
Miauuu
Gato@75a1cd57
Gato@75a1cd57
Miauuu
```

Sobrescrevendo métodos de Object

```java
public class Animal{
    private int idade;
    private String nome;
    public Animal() {
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void emitirSom() {
        System.out.println("Som genérico");
    }
    public String toString() {
        return getNome();
    }
}
```

```java
public class HerancaArrays {

    public static void main(String[] args) {
        Animal [] corrida = {new Gato(), new Cobra(), new Cobra(), new Gato(), new Gato()};
        corrida[0].setNome("Felix");
        corrida[1].setNome("C1");
        corrida[1].setNome("C2");
        corrida[3].setNome("Tom");
        corrida[4].setNome("Mingau");
        for (int i = 0; i < corrida.length; i++) {
            Animal a = corrida[i];
            System.out.println(a); //toString() sendo invocado implicitamente
            a.emitirSom();
        }
    }
}
```

Problems Javadoc Declaration Console

<terminated> HerancaArrays [Java Application] /Users/gustavo/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2.v20200815-0932/jre/bin/java

```
Felix
Miauuu
C2
ssssss
null
ssssss
Tom
Miauuu
Mingau
Miauuu
```

Sedimentação

- Estudar o capítulo 9 da apostila da Caelum (FJ11)
- https://www.caelum.com.br/apostila/apostila-java-orientacao-objetos.pdf