

ARA0066 - PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA



ARA0075

Aula - 05



Introdução - Polimorfismo

Polimorfismo

O polimorfismo deriva da palavra polimorfo, que significa multiforme, ou que pode variar a forma. Para a POO, polimorfismo é a habilidade de objetos de classes diferentes responderem a mesma mensagem de diferentes maneiras. Ou seja, várias formas de responder à mesma mensagem. Veja a figura a seguir para entender onde se localiza o pilar do polimorfismo dentro da Programação Orientada a Objetos.



Dicionário

polimorfismo



polimorfismo

substantivo masculino

1. qualidade ou estado de ser capaz de assumir diferentes formas.

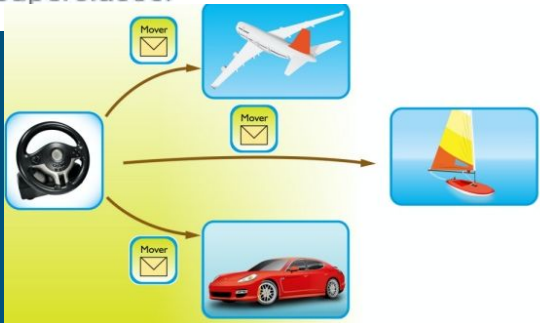
Antes de aplicarmos esse conceito ao conceito de Programação Orientada a Objetos, segue na figura abaixo o conceito extraído de <https://dicionario.priberam.org/polimorfismo>:

Polimorfismo

Em OO, **polimorfismo** é a capacidade de um objeto se comportar de diferentes maneiras.

Mais uma vez convém destacar que se trata de um princípio de OO que Java implementa, assim como várias linguagens OO.

O polimorfismo pode se expressar de diversas maneiras. A sobrecarga de função, assim como a herança, são formas de dar ao objeto uma capacidade polimórfica. No caso da herança, o polimorfismo surge justamente porque um objeto pode se comportar também como definido na superclasse.

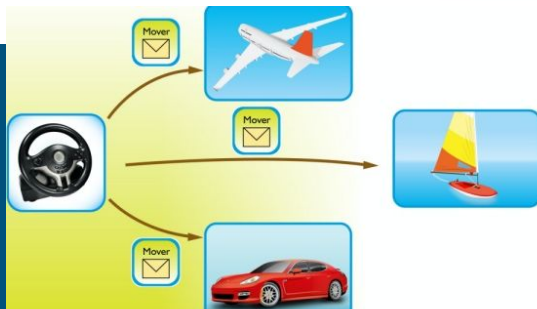


Polimorfismo

Todo objeto que possui uma superclasse tem capacidade de ser polimórfico. Por essa razão, todo objeto em Java é polimórfico, mesmo que ele não estenda explicitamente outra classe. A justificativa, como já dissemos, é que toda classe em Java descende direta ou indiretamente da classe "Object".

O polimorfismo permite o desenvolvimento de códigos facilmente extensíveis, pois novas classes podem ser adicionadas com baixo impacto para o restante do software. Basta que as novas classes sejam derivadas daquelas que implementam comportamentos gerais, como no caso da classe "Pessoa".

Essas novas classes podem especializar os comportamentos da superclasse, isto é, alterar a sua implementação para refletir sua especificidade, e isso não impactará as demais partes do programa que se valem dos comportamentos da superclasse.



**POLIMORFISMO
EM JAVA**

Métodos e algoritmos polimórficos
aplicados a jogos de computador

7ª edição (2022)

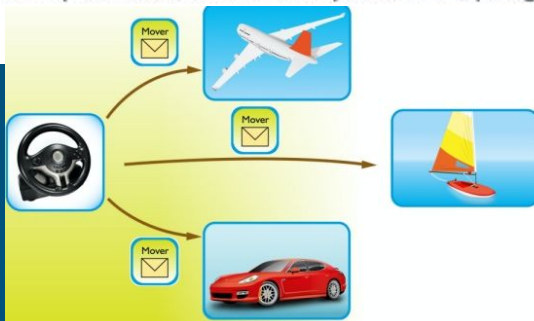


Polimorfismo

Todo objeto que possui uma superclasse tem capacidade de ser polimórfico. Por essa razão, todo objeto em Java é polimórfico, mesmo que ele não estenda explicitamente outra classe. A justificativa, como já dissemos, é que toda classe em Java descende direta ou indiretamente da classe "Object".

O polimorfismo permite o desenvolvimento de códigos facilmente extensíveis, pois novas classes podem ser adicionadas com baixo impacto para o restante do software. Basta que as novas classes sejam derivadas daquelas que implementam comportamentos gerais, como no caso da classe "Pessoa".

Essas novas classes podem especializar os comportamentos da superclasse, isto é, alterar a sua implementação para refletir sua especificidade, e isso não impactará as demais partes do programa que se valem dos comportamentos da superclasse.



Polimorfismo - Entendendo na prática I

Dizer que uma **Pessoa** É **UMA PessoaFisica** está errado, porque ela pode também ser uma **PessoaJuridica**.

Quando trabalhamos com uma variável do tipo `Pessoa` que é uma super classe, podemos fazer esta variável receber um objeto do tipo `PessoaFisica` ou `PessoaJuridica`, por exemplo:

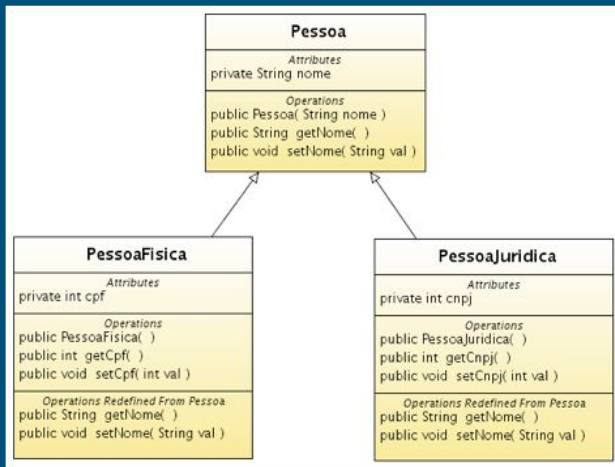
```
Pessoa fisica = new PessoaFisica();  
Pessoa juridica = new PessoaJuridica();
```

Com isso, podemos dizer que **polimorfismo** é a capacidade de um objeto ser referenciado de diversas formas diferentes e com isso realizar as mesmas tarefas (ou chamadas de métodos) de diferentes formas.



Polimorfismo - Entendendo na prática II

Um exemplo do uso do polimorfismo utilizando a classe `Pessoa`, seria todas as subclasses sobrescreverem o método `public String getNome()`.



```
package material.polimorfismo;
```

```
/**
 * Classe utilizada para representar uma Pessoa.
 */
public class Pessoa {
    private String nome;

    public String getNome() {
        return nome;
    }

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

POLIMORFISMO
EM JAVA

Métodos e algoritmos polimórficos
aplicados a jogos de computador

7 ed. (2012)

Polimorfismo - Entendendo na prática II

Um exemplo do uso do polimorfismo utilizando a classe `Pessoa`, seria todas as subclasses sobrescreverem o método `public String getNome()`.

A subclasse `PessoaFisica` sobrescreve o método `public String getNome()` e retorna a seguinte frase: “Pessoa Fisica: nomePessoa – CPF: cpfPessoa”.

```
package material.polimorfismo;


/**
 * Classe utilizada para representar uma Pessoa Fisica
 * que É UMA subclasse de Pessoa.
 */
public class PessoaFisica extends Pessoa {
    private long cpf;

    public PessoaFisica() {
    }

    public long getCpf() {
        return cpf;
    }

    public void setCpf(long cpf) {
        this.cpf = cpf;
    }

    public String getNome() {
        return "Pessoa Fisica: " + super.getNome() + " - CPF: " + this.getCpf();
    }
}
```



Polimorfismo - Entendendo na prática II

A subclasse `PessoaJuridica` sobreescreve o método `public String getNome()` e retorna a seguinte frase: "Pessoa Juridica: nomePessoa – CNPJ: cnpjPessoa".

Desta maneira, independentemente do nosso objeto `PessoaFisica` e `PessoaJuridica` ter sido atribuído a uma referencia para `Pessoa`, quando chamamos o método `public String getNome()` de ambas variáveis, temos a seguinte saída:

```
Pessoa Fisica: Cristiano - CPF: 0  
Pessoa Juridica: Rafael - CNPJ: 0
```



```
package material.polimorfismo;  
  
/**  
 * Classe utilizada para representar uma Pessoa Fisica  
 * que É UMA subclasse de Pessoa.  
 */  
public class PessoaJuridica extends Pessoa {  
    private long cnpj;  
  
    public PessoaJuridica() {  
    }  
  
    public long getCnpj() {  
        return cnpj;  
    }  
  
    public void setCnpj(long cnpj) {  
        this.cnpj = cnpj;  
    }  
  
    public String getNome() {  
        return "Pessoa Juridica: " + super.getNome() + " - CNPJ: " + this.getCnpj();  
    }  
}
```



Polimorfismo - Entendendo na prática II

Mesmo as variáveis sendo do tipo `Pessoa`, o método `public String getNome()` foi chamado da classe `PessoaFisica` e `PessoaJuridica`, porque durante a execução do programa, a JVM percebe que a variável `fisica` está guardando um objeto do tipo `PessoaFisica`, e a variável `juridica` está guardando um objeto do tipo `PessoaJuridica`.

```
public static void main(String[] args) {  
    Pessoa fisica = new PessoaFisica();  
    fisica.setNome("Cristiano");  
    fisica.setCpf(12345678901L);  
}
```

Note que neste exemplo apenas atribuímos o valor do nome da `Pessoa`, não informamos qual o CPF ou CNPJ da pessoa, se tentarmos utilizar a variável do tipo `Pessoa` para atribuir o CPF através do método `public void setCpf(long cpf)` teremos um erro de compilação, pois somente a classe `PessoaFisica` possui este método:

Durante a compilação teremos o seguinte erro informando que a classe `material.polimorfismo.Pessoa` não possui o método `public void setCpf(long cpf)`:



```
C:\>javac material\polimorfismo\TestePessoa.java  
material\polimorfismo\TestePessoa.java:12: cannot find symbol  
symbol : method setCpf(long)  
location: class material.polimorfismo.Pessoa  
    fisica.setCpf(12345678901L);  
    ^  
1 error
```

Polimorfismo - Entendendo na prática II

Logo a forma correta de declaração para a correta herança do método é:

```
*/
public class Main {
    public static void main(String[] args) {
        PessoaFisica fisica = new PessoaFisica();
        fisica.setNome("Cristiano");
        fisica.setCpf(12345);

        Pessoa juridica = new PessoaJuridica();
        juridica.setNome("Rafael");

        Pessoa[] pessoas = new Pessoa[2];
        pessoas[0] = fisica;
        pessoas[1] = juridica;

        for(Pessoa pessoa : pessoas) {
            System.out.println(pessoa.getNome());
        }
    }
}
```

Classes abstratas

Pode-se dizer que as classes abstratas servem como “modelo” para outras classes que dela herdem, não podendo ser instanciada por si só. Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe. Os métodos da classe abstrata devem então serem sobrescritos nas classes filhas.

Por exemplo, é definido que a classe “Animal” seja herdada pelas subclasses “Gato”, “Cachorro”, “Cavalo”, mas ela mesma nunca pode ser instanciada.

```
abstract class Conta {  
  
    private double saldo;  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public double getSaldo() {  
        return saldo;  
    }  
  
    public abstract void imprimeExtrato();  
  
}
```

Fonte:

<https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/>

Classes abstratas

No exemplo da Listagem 5, o método “`imprimeExtrato()`” tem uma annotation conhecida como `@Override`, significando que estamos sobrescrevendo o método da superclasse. Entende-se em que na classe abstrata “Conta” os métodos que são abstratos têm um comportamento diferente, por isso não possuem corpo. Ou seja, as subclasses que estão herdando precisam desse método mas não de forma genérica, aonde permite inserir as particularidades de cada subclasse.

```
public class TestaConta {  
    public static void main(String[] args) {  
        Conta cp = new ContaPoupanca();  
        cp.setSaldo(2121);  
        cp.imprimeExtrato();  
    }  
}
```

```
import java.text.SimpleDateFormat;  
import java.util.Date;
```

```
public class ContaPoupanca extends Conta {
```

```
    @Override
```

```
    public void imprimeExtrato() {
```

```
        System.out.println("### Extrato da Conta ###");
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/aaaa HH:mm:ss");
```

```
        Date date = new Date();
```

```
        System.out.println("Saldo: "+this.getSaldo());
```

```
        System.out.println("Data: "+sdf.format(date));
```

```
    }
```

```
}
```

Fonte:

<https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/>

Atividade

Desenvolva um sistema de de cadastro de cliente para uma empresa de prestação de serviços, que possui tanto clientes pessoa física quanto jurídica.

Para essa construção a superclasse deve ser “Pessoa”, a ser especializada em física e jurídica.

Cada classe deve ser implementada com métodos específicos, porém os métodos ligados aos atributos gerais da classe pessoa, precisam ser especializados. Além disso uma dessas classes deve ser do tipo “Abstrata”.

O cadastro deve ser realizado pelo usuário, por meio do teclado, desse modo é necessário o uso de um menu para selecionar o número e o tipo de cliente a ser cadastrado.

Antes de começar utilize o Tema 2 Módulo 4, para revisar os comando básicos de Java.