

Polimorfismo e Exemplos

Leandro Ungari Cayres¹, Danilo Medeiros Eler²

Universidade Estadual Paulista

leandro.ungari@unesp.br¹, danilo.eler@unesp.br²

13 de fevereiro de 2019

Visão Geral

1 Conceitos

- Definição
- Exemplo inicial

2 Tipos de Polimorfismo

- Polimorfismo de Inclusão
- Polimorfismo Paramétrico
- Sobreposição
- Sobrecarga

3 Outros Exemplos

- Pagamento de Compras

Introdução

Neste ponto da Orientação a Objetos, foram introduzidos dois pilares desse paradigma, o Encapsulamento e a Herança, enquanto o primeiro permite a implementação de componentes independentes, o segundo possibilita a reutilização hierárquica desses.

Polimorfismo

Definição

Ter muitas formas. Em termos de programação, muitas formas significa que um único nome pode representar um código diferente, selecionado por algum elemento automático. Assim, o polimorfismo permite que um único nome expresse muitos comportamentos diferentes [1].

Características

O polimorfismo atende cada um dos objetivos da orientação a objetos, permitindo a elaboração de softwares com as seguintes características [1]:

- Natural

O polimorfismo permite que modele o mundo de forma mais natural, trabalhando em nível mais genérico e conceitual.

- Confiável

O polimorfismo resulta em código confiável pois simplifica os casos, eliminando situações especiais e isolando o código, assim reduz a chance de introduzir defeitos.

Características

■ Reutilizável

O polimorfismo permite reaproveitar implementadas já realizadas, somente adequando conversa a interface utilizada, sempre precisar conhecer detalhes específicos.

■ Manutenível

O polimorfismo resulta em menos código, por consequência menos código precisa ser mantido, ou seja, menos trabalho.

Características

■ Extensível

O polimorfismo permite que novos tipos sejam adicionados sem afetar as demais partes do sistema, assim a integração é facilitada.

■ Oportuno

O polimorfismo permite escrever menos código, e consequentemente, pode distribuí-lo mais cedo.

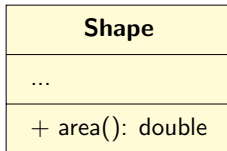
Cálculo de Área

Uma aplicação está sendo desenvolvida para o cálculo da áreas, requerindo a necessidade implementação de algumas figuras geométricas.

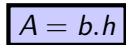
Como podemos definir o cálculo da área para essas figuras?

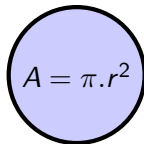
Cálculo de Área

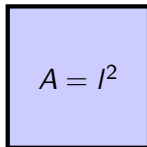
Um diagrama de classe que projetaria esta aplicação.

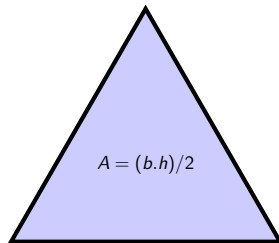


Cálculo de Área

A light blue rectangle with a black border.
$$A = b.h$$

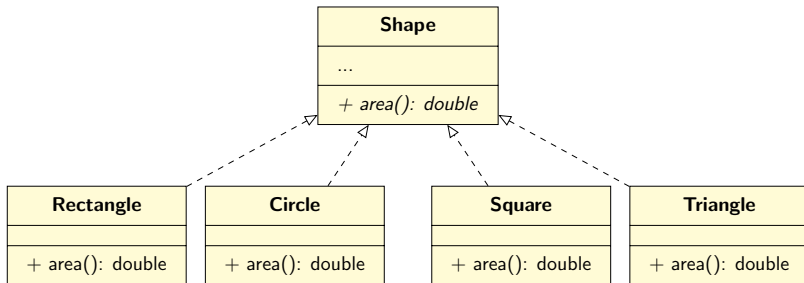
A light blue circle with a black border.
$$A = \pi.r^2$$

A light blue square with a black border.
$$A = l^2$$

A light blue triangle with a black border.
$$A = (b.h)/2$$

Cálculo de Área

Um diagrama de classe que projetaria esta aplicação.



Cálculo de Área – Implementação

```
public class Shape {  
    ...  
    public abstract double area();  
}  
  
public class Rectangle extends Shape {  
  
    private double base;  
    private double height;  
    ...  
    public abstract double area() {  
        return this.base*this.height;  
    }  
}
```

Cálculo de Área – Implementação

```
public class Circle extends Shape {  
  
    private double radius;  
    ...  
    public abstract double area() {  
        return Math.PI*Math.pow(this.radius, 2);  
    }  
}  
  
public class Square extends Shape {  
  
    private double side;  
    ...
```

Cálculo de Área – Implementação

```
...
public abstract double area() {
    return Math.pow(this.side, 2);
}
}

public class Triangle extends Shape {

    private double base;
    private double height;
    ...
    public abstract double area() {
        return this.base*this.height/2;
    }
}
```

Tipos de Polimorfismo

Categorias

O Polimorfismo pode ser aplicado de diversos modos, conforme a necessidade de cada implementação [1]. A seguir são apresentados quatros principais categorias dessa abordagem:

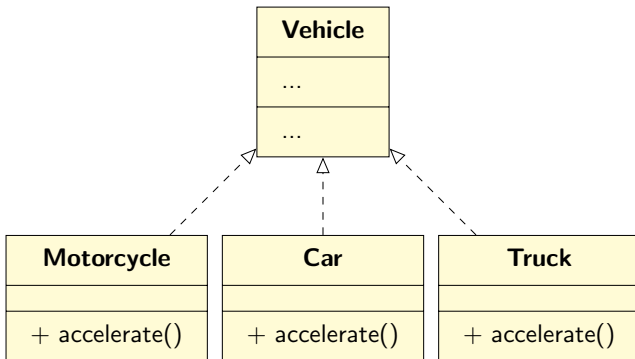
- Polimorfismo de Inclusão
- Polimorfismo Paramétrico
- Sobreposição
- Sobrecarga

Definição

Conceito

O polimorfismo de inclusão, também conhecido como polimorfismo puro, permite que você trate objetos relacionados genericamente.

Exemplo



Exemplo

```
public void accelerateVehicle(Motorcycle obj){  
    obj.accelerate();  
}
```

```
public void accelerateVehicle(Car obj){  
    obj.accelerate();  
}
```

```
public void accelerateVehicle(Truck obj){  
    obj.accelerate();  
}
```

Exemplo

Todos os métodos anteriores são equivalentes a esse:

```
public void accelerateVehicle(Vehicle obj){  
    obj.accelerate();  
}
```

Definição

Conceito

O polimorfismo paramétrico que você crie métodos e tipos genéricos. Essa categoria pode ser aplicada em tipos e métodos, na primeira os tipos de parâmetros e retornos podem ser generalizados, enquanto no segundo, as referências de tipos dos métodos são omitidos até o momento da execução.

Exemplo

```
//Polimorfismo parametrico de Tipo
public class Queue<T> {

    void enqueue(T element) { ... }

    T dequeue() { ... }

    boolean isEmpty() { ... }

}
```

Exemplo

```
//Metodos de adicao de elementos
int add(int a, int b) { ... }
double add(double a, double b) { ... }
int[][] add(int[][] a, int[][] b) { ... }

//Polimorfismo parametrico de Metodo
T add(T a, T b) { ... }
```

Definição

Conceito

O polimorfismo de sobreposição é um tipo importante, no qual o comportamento em uma hierarquia é sobreposto por um comportamento mais específico

Exemplo

Uma classe **Shape** é definida, da qual derivam outras duas classes **Square** e **Triangle**.

```
public abstract class Shape {  
  
    public abstract double area();  
  
}
```

Exemplo

```
@Override
public abstract class Square extends Geometric {

    public double area() {
        return this.side*this.side;
    }
}

@Override
public abstract class Triangle extends Geometric {

    public double area() {
        return this.base*this.height/2;
    }
}
```

Definição

Conceito

O polimorfismo de sobrecarga, também conhecido como polimorfismo *ad-hoc*, permite que o nome do método seja o mesmo para diferentes métodos, diferindo apenas na quantidade de parâmetros e os tipo deles.

Exemplo

```
public class MyOwnDB {  
  
    //It returns all items  
    public List<Item> search() { ... }  
  
    //It returns all items which are filtered by condition  
    public List<Item> search(String condition) { ... }  
  
    //... and ordered by attribute  
    public List<Item> search(String condition, String  
        orderBy) { ... }  
  
    //It returns limited number of items which are  
        filtered by condition and ordered by attribute  
    public List<Item> search(String condition, String  
        orderBy) { ... }  
}
```

Armadilhas Polimórficas

A utilização do polimorfismo permite muitas facilidades, porém a utilização desses recursos de modo desordenado pode incorrer em diversos problemas para a arquitetura do sistema.

- Mover comportamentos para cima da hierarquia
Mover métodos acima pode dar capacidades a classes em que estas não deveriam ter.

Armadilhas Polimórficas

■ Sobrecarga de desempenho

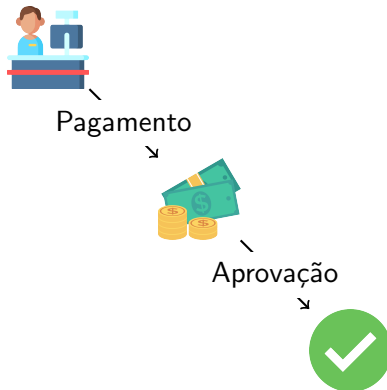
A generalização de tipos de parâmetros e retorno de métodos traz perda de desempenho em tempo de execução, em que a verificação de tipos, que era feita estaticamente, será feita dinamicamente, demandando processamento.

■ Perda de comportamento

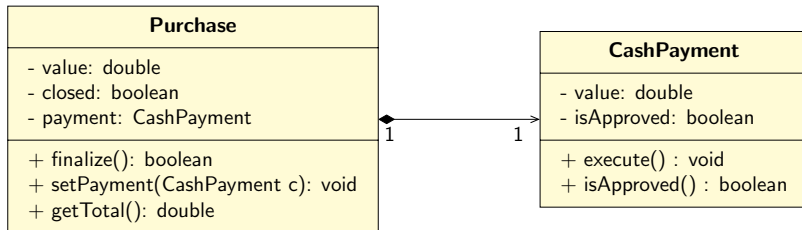
Na generalização de tipos específicos, um método ou atributo pertencente a uma subclasse se torna inacessível.

Outros Exemplos

Uma pequena empresa aceitava inicialmente somente pagamentos em dinheiro, desse modo, seu sistema de caixa fora projetado para tal propósito.



Representação



Implementação

```
public class Purchase {  
    ...  
  
    public boolean finalize() {  
        ...  
        //type CashPayment  
        this.payment.execute();  
    }  
}
```

```
Purchase purchase = new Purchase();  
CashPayment cash = new CashPayment(purchase.getTotal());  
  
purchase.setPayment(cash);  
purchase.finalize();
```

Exemplo I

Com o crescimento da empresa e também solicitação dos usuários, novas formas de pagamento devem ser adicionadas.

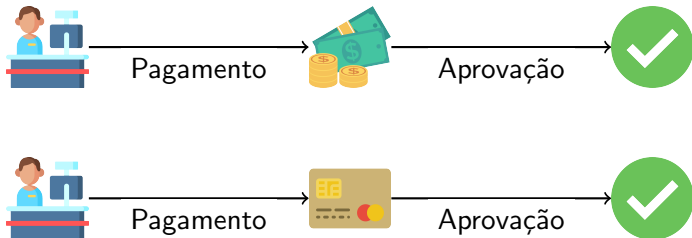


Qual é a melhor estratégia a ser desenvolvida na implementação do sistema?

Opção I



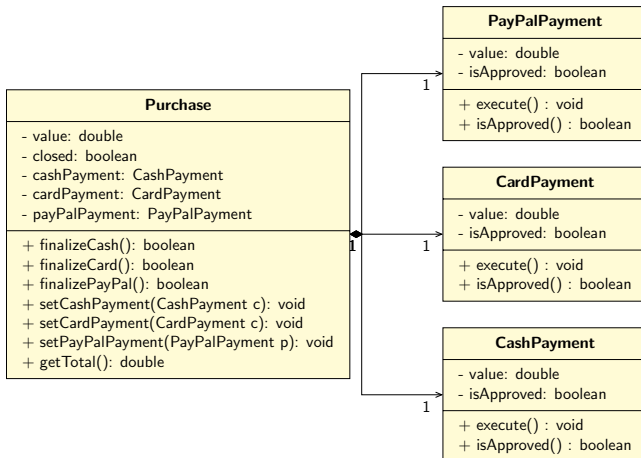
Opção I



Opção I



Representação



Implementação

```
public class Purchase {  
    ...  
  
    public boolean finalize() {  
        ...  
        //type CashPayment  
        this.cashPayment.execute();  
    }  
}
```

```
Purchase purchase = new Purchase();  
CashPayment cash = new CashPayment(purchase.getTotal());  
  
purchase.setCashPayment(cash);  
purchase.finalizeCash();
```


Implementação

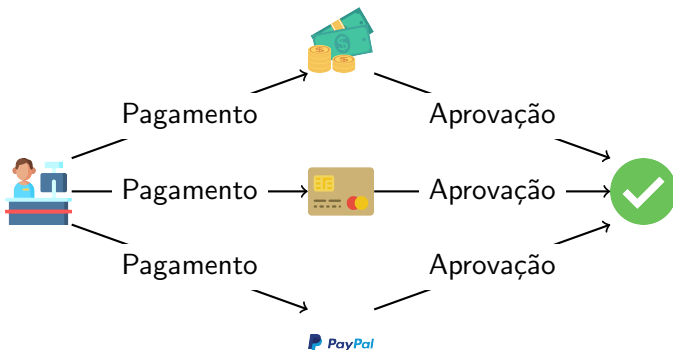
```
public class Purchase {  
    ...  
  
    public boolean finalize() {  
        ...  
        //type CardPayment  
        this.cardPayment.execute();  
    }  
}
```

```
Purchase purchase = new Purchase();  
CardPayment card = new CardPayment(purchase.getTotal());  
  
purchase.setCardPayment(card);  
purchase.finalizeCard();
```

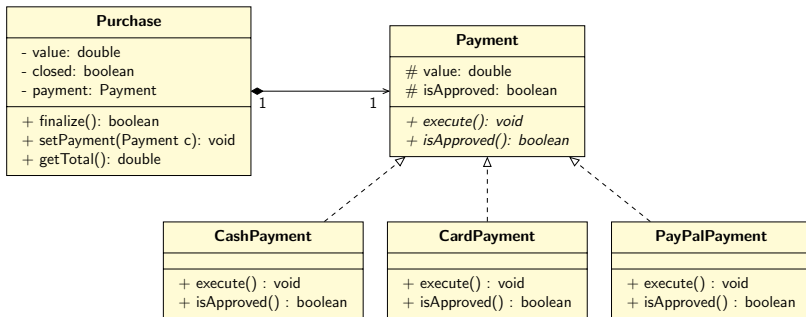
Implementação

```
public class Purchase {  
    ...  
  
    public boolean finalize() {  
        ...  
        //type PayPalPayment  
        this.payPalPayment.execute();  
    }  
}  
  
Purchase purchase = new Purchase();  
PayPalPayment payPal = new  
    PayPalPayment(purchase.getTotal());  
  
purchase.setPayPalPayment(payPal);  
purchase.finalize();
```

Opção II



Representação



Implementação

```
public class Purchase {  
  
    public boolean finalize() {  
        ...  
        //type Payment  
        this.payment.execute();  
    }  
}
```

```
Purchase purchase = new Purchase();  
Payment payment = new CardPayment(purchase.getTotal());  
                //new CashPayment  
                //new PayPalPayment  
  
purchase.setPayment(payment);  
purchase.finalize();
```

Referência Bibliográfica I



A. Sintes, *Aprenda programação orientada a objetos em 21 dias.*

Makron Books, 2002.