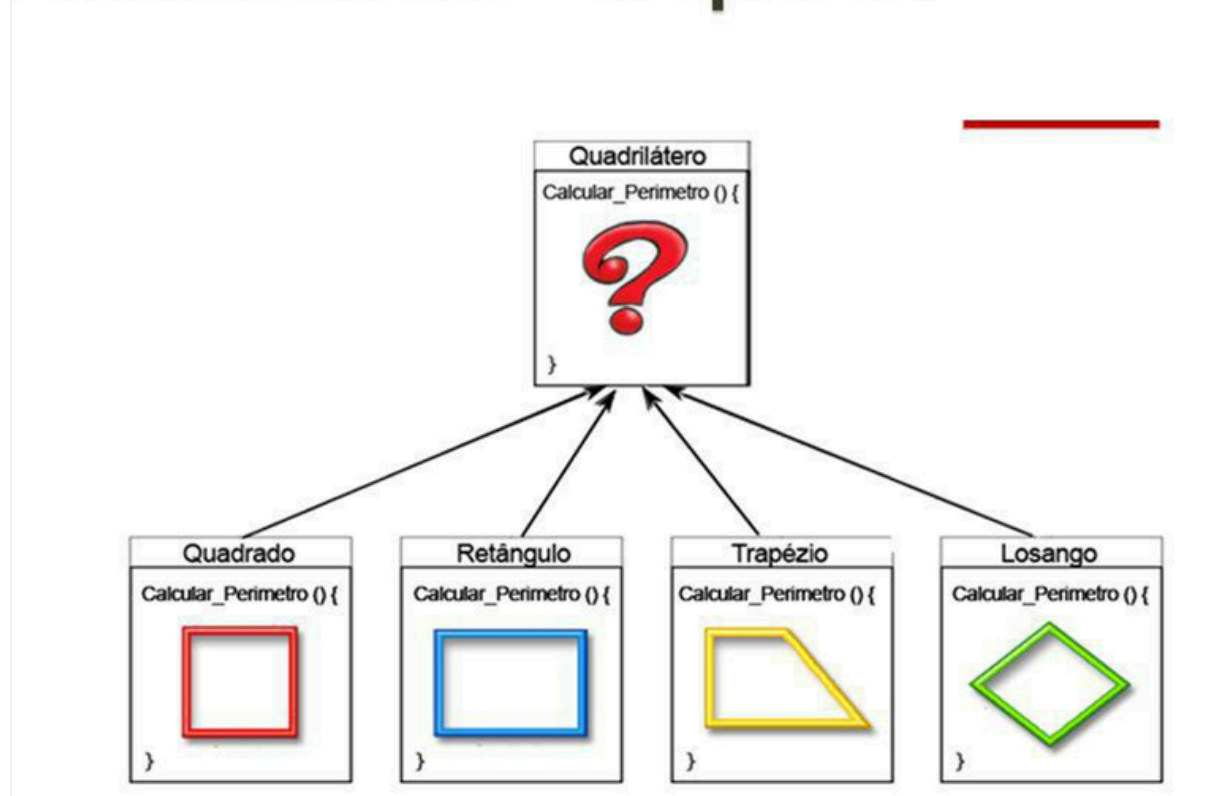


# Polimorfismo

Um mesmo comportamento, de várias maneiras.

Podemos observar no contexto de **Abstração e Herança**, que conseguimos criar uma singularidade estrutural de nossos elementos. Isso quer dizer que, qualquer classe que deseja representar um serviço de mensagens, basta estender a classe `ServicoMensagemInstantanea` e implementar, os respectivos métodos *abstratos*. O que vale reforçar aqui é, cada classe terá a mesma ação, executando procedimentos de maneira especializada.

## Polimorfismo – O que é?



Este é o resultado do que denominamos como, Polimorfismo. Veja o exemplo abaixo:

```
public class ComputadorPedrinho {
    public static void main(String[] args) {

        ServicoMensagemInstantanea smi = null;

        /*
            NÃO SE SABE QUAL APP
            MAS QUALQUER UM DEVERÁ ENVIAR E RECEBER MENSAGEM
        */
        String appEscolhido="???";

        if(appEscolhido.equals("msn"))
            smi = new MSNMessenger();
        else if(appEscolhido.equals("fbm"))
            smi = new FacebookMessenger();
        else if(appEscolhido.equals("tlg"))
            smi = new Telegram();

        smi.enviarMensagem();
        smi.receberMensagem();
    }
}
```

**i** Para concluirmos a compreensão, Polimorfismo permite que as classes mais abstratas, determinem ações uniformes, para que cada classe filha concreta, implemente os comportamentos de forma específica.

## Modificador protected

Vamos para uma retrospectiva, quanto ao requisito do nosso sistema de mensagens instantâneas, desde a etapa de encapsulamento.

O nosso requisito, solicita que além de Enviar e Receber Mensagens, precisamos validar se o aplicativo está conectado a internet ( `validarConectadoInternet` ) e salvar o histórico de cada mensagem ( `salvarHistoricoMensagem` ).

Sabemos que cada aplicativo, costuma salvar as mensagens em seus respectivos servidores cloud, mas e quanto validar se está conectado a internet? Não poderia ser um mecanismo comum a todos ? Logo, qualquer classe filha, de **ServicoMensagemInstantanea** poderia desfrutar através de herança, esta funcionalidade.

**i** Mas fica a reflexão do que já aprendemos sobre visibilidade de recursos: Com o modificador `** private` \*\*somente a classe conhece a implementação, quanto que o modificador `** public` \*\*todos passarão a conhecer. Mas gostaríamos que, somente as classes filhas soubessem. Bem, é aí que entra o modificador `protected` .j

```
public abstract class ServicoMensagemInstantanea {  
  
    public abstract void enviarMensagem();  
    public abstract void receberMensagem();  
  
    //mais um método que todos os filhos deverão implementar  
    public abstract void salvarHistoricoMensagem();  
  
    //somente os filhos conhecem este método  
    protected void validarConectadoInternet() {  
        System.out.println("Validando se está conectado a internet");  
    }  
}
```

## Referências



Polimorfismo (ciência da computação)  
Wikipedia



Previous  
Abstração

Next  
Interface

Last updated 1 year ago