**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA**

**ESCOLA TÉCNICA ESTADUAL DA ZONA LESTE**

**Técnico em Desenvolvimento de Sistemas**

**Andrey Ramos Caldas,**

**Gianlucca Silva Campana Ferreira**

**Blacal Elétrica**

**São Paulo**

**2018**

**Andrey Ramos Caldas,**

**Gianlucca Silva Campana Ferreira**

**Blacal Elétrica**

Trabalho de Conclusão de Curso apresentado a ETEC da Zona Leste, para a disciplina de Desenvolvimento de Trabalho de Conclusão de Curso, administrada pelo Professor Rogerio, como requisito final para obtenção do título de Técnico em Desenvolvimento de Sistemas.

**São Paulo**

**2018**

**AGRADECIMENTOS**

**EPÍGRAFE**

**RESUMO**

**ABSTRACT**

**LISTA DE ILUSTRAÇÕES**

[**Figura 1: Mobile Electrician** 13](#_Toc21545814)

[**Figura 2: DIALux Mobile** 14](#_Toc21545815)

[**Figura 3: Softlux** 15](#_Toc21545816)

[**Figura 4: Programa em Java** 16](#_Toc21545817)

[**Figura 5: Nomes válidos:** 16](#_Toc21545818)

[**Figura 6: Nomes inválidos** 17](#_Toc21545819)

[**Figura 7: Mostra um exemplo de uma classe criada em Java com seus atributos.** 17](#_Toc21545820)

[**Figura 8: instancia de um objeto** 18](#_Toc21545821)

[**Figura 9: Exemplo de diagrama de caso de uso** 20](#_Toc21545822)

[**Figura 10: Exemplo de diagrama de atividade** 21](#_Toc21545823)

[**Figura 11: diagrama de Classe** 23](#_Toc21545824)

[**Figura 12: Detalhamento das assinaturas das operações** 24](#_Toc21545825)

[**Figura 13: detalhamento dos Atributos** 24](#_Toc21545826)

[**Figura 14: Generalização ou Especialização** 25](#_Toc21545827)

[**Figura 15: Diagrama de sequência** 27](#_Toc21545828)

[**Figura 16: Chave/Valor** 28](#_Toc21545829)

[**Figura 17: Exemplo de orientado a objetos** 29](#_Toc21545830)

[**Figura 18: Exemplo de orientado a colunas.** 29](#_Toc21545831)

[**Figura 19: Exemplo de Grafos** 30](#_Toc21545832)

**LISTA DE ABREVIATURAS E SIGLAS**

**UML**

**NBR 5410**

**OMT**

**OOSE**

**NoSql**

**SUMÁRIO**

[1 Introdução 12](#_Toc21542444)

[2 Referencial teórico 13](#_Toc21542445)

[2.1 Aplicativos de instalações elétricas 13](#_Toc21542446)

[2.2 Java 15](#_Toc21542447)

[2.3 UML 18](#_Toc21542448)

[2.3.1 Levantamento de requisitos 19](#_Toc21542449)

[2.3.2 Diagramas de Caso de Uso 20](#_Toc21542450)

[2.3.3 Diagrama de atividades 21](#_Toc21542451)

[2.3.4 Diagrama de classes 22](#_Toc21542452)

[2.3.5 Diagrama de sequência 25](#_Toc21542453)

[2.4 NoSql 27](#_Toc21542454)

[2.4.1 Chave/Valor 28](#_Toc21542455)

[2.4.2 Orientado a objetos 28](#_Toc21542456)

[2.4.3 Orientado a colunas 29](#_Toc21542457)

[2.4.4 Banco de dados de grafos 30](#_Toc21542458)

[3 NBR 5410 30](#_Toc21542459)

[3.1.1 Iluminação 32](#_Toc21542460)

[3.1.2 Pontos de tomadas 32](#_Toc21542461)

[4 Desenvolvimento 34](#_Toc21542462)

[5 Conclusão 35](#_Toc21542463)

[Referências 36](#_Toc21542464)

# Introdução

Instalações elétricas de tomadas e luminárias, necessitam de cálculos conforme NBR 5410, que podem ser demorados e difíceis para se fazer sem auxílio tecnológico. No mundo contemporâneo, o diferencial é agilidade e a exatidão, para fazer os cálculos mais rápidos e sem preocupação em saber se eles estão corretos. Como um aplicativo ajudaria os eletricistas no seu trabalho?

Na tecnologia, se tem a possibilidade de implementar ferramentas para auxiliar no seu dia-a-dia profissional. Ferramentas com essa finalidade, tende a facilitar seu trabalho, poupando tempo e diminuindo a chance de ocorrer erros que podem ser fatais.

O desenvolvimento de um aplicativo Android que auxilie nos cálculos de tomadas e luminárias em uma instalação, seguindo NBR 5410, com o máximo de precisão. E se mostrar como uma ajuda na melhora e velocidade de trabalho.

Com as pesquisas bibliográficas, estudo de Unified Moling Language (UML), NoSql, Java, foi feito um aplicativo que recebe dados do usuário e que serão processados e retornaram os resultados dos cálculos conforme a NBR 5410, com prósito de melhorar a eficiência de uma instalação elétrica.

# Referencial teórico

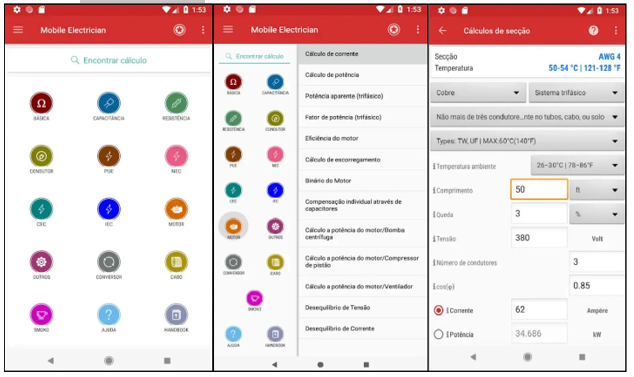
Neste capítulo contêm a base teórica das tecnologias necessárias para o desenvolvimento do projeto de pesquisa da aplicação mobile de instalações elétricas

## Aplicativos de instalações elétricas

Existem alguns aplicativos ou softwares no mundo da elétrica que auxiliam no mundo dos eletricistas e que tomamos como base para fazer o nosso.

O aplicativo Mobile Electrician vem com umas enormes variedades de opções de funcionalidades que são voltadas para os cálculos elétricos. Por exemplo, temos a seção de cálculos de bitolas de cabos, onde você vai encontrar várias opções para descobrir como fazer os cálculos corretos para a instalação elétrica. Pode ser encontrado também os cálculos para carga máxima do circuito, corrente alternada, capacidade de bateria, entre outros. E eles têm a funcionalidade de usar off-line, sem depender de internet.

**Figura 1: Mobile Electrician**



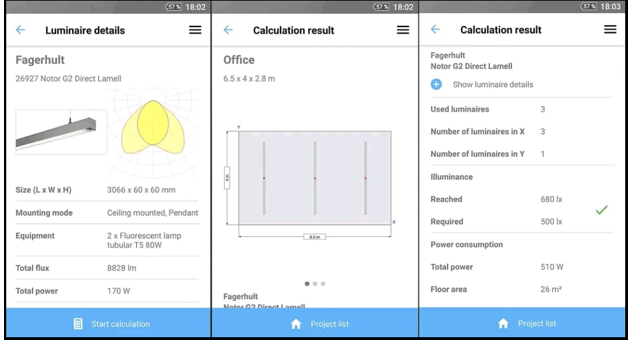
Fonte:?????

Um concorrente do Mobile Electrician seria um aplicativo chamado Electrical Technology, onde se consegue fazer cálculos de circuitos, dimensionamento de cabos, queda de tensão, resistência e mais algumas outras funções. Ele também pode ser usado off-line, sem o uso de internet. Mas ele é totalmente em inglês, onde limita algumas pessoas a usarem o aplicativo.

Também temos o iCircuit, um outro aplicativo para mais fins educativos, ou seja, permite que o usuário consiga testar algumas ligações de circuitos, onde se aprende com os erros e acertos sem correr riscos nas instalações elétricas. Neste aplicativo contem simuladores que permitem lidar com circuitos analógicos e digitais, apresentando uma análise completa em tempo real.

DIALux Mobile é um aplicativo desenvolvido pela empresa DIAL GmbH em julho de 2018, onde se assemelha mais com a nossa linha de pensamento para cálculos de instalações elétricas. Neste aplicativo tem função principal planejar a iluminação de um espaço interior com produtos reais, onde se pode calcular os lumens e obter também as distribuições de luminárias num ambiente determinado pelos usuários, podendo também escolher o tipo de luminárias para melhor atender às necessidades dos usuários. A DIALux tem uma aplicação para desktop, onde é a ferramenta mais popular do mundo quando se fala em planejamento de luz. Onde pode-se planejar profissionalmente, calcular e visualizar a iluminação.

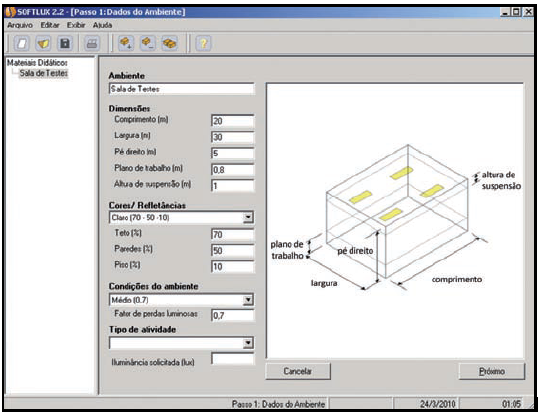
**Figura 2: DIALux Mobile**



Fonte:?????

Existem softwares que servem para agilizar o processo de cálculos de instalações de luminárias, um dos principais concorrentes do DIALux é o SoftLux, desenvolvido pela empresa ITAIM Iluminação, um software para desktop desenvolvido para designers.

**Figura 3: Softlux**



Fonte:????

## Java

Segundo Furgeri (2015), em 1995, a SUN anunciou o JAVA, não como só mais uma linguagem de programação e sim como uma nova plataforma de desenvolvimento. Desse jeito o Java começou a ser muito utilizada em criações de páginas para World Wide Web (www), criando uma produção de conteúdo em intervalo dinâmico.

Como diz Claro e Mangueira (2008), Java é uma linguagem de programação orientada a objetos, capaz de criar um aplicativo para desktop, aplicações comerciais, software robustos, mais completos e independentes e aplicações para web.

Segundo Fugeri(2015), a linguagem Java possui as seguintes características:

* Orientação a objetos: A maioria das linguagens permitem trabalhar dessa forma. Orientação a objetos é uma prática que torna possível a criação de um software a partir de gerações de objetos que se falam entre si.
* Portabilidade: Java é uma linguagem que permite ser utilizada em diferentes tipos de plataformas sem a necessidade a mudança de código. Com isso permite que um programa em Java possa usado em qualquer sistema operacional.
* Multithreading: São linhas de execução que consegue fazer que mais de um evento aconteça simultaneamente de um programa

Conforme Paul e Harvey (2016), uma aplicação Java é um programa de computador que é inicializado quando se utiliza comando java para carregar Java Virtual Machine (JVM).

**Figura 4: Programa em Java**



Fonte: Paul e Harvey, 2016.

Como diz Thiago e Normandes(2014), no Java as variáveis podem conter letras, \_(underscore) e $(dólar), mas não pode ser iniciadas por um número e palavras reservadas.

**Figura 5: Nomes válidos:**



Fontes: Segundo Thiago e Normandes, 2014

**Figura 6: Nomes inválidos**



Fontes: Segundo Thiago e Normandes, 2014

Classe, é uma descrição de um conjunto de entidades (reais ou abstratas) com mesmo tipo, características e comportamento. São as classes que definem a estrutura e o comportamento dos objetos de um determinado tipo, diz Thiago e Noemandes (2014).

**Figura 7: Mostra um exemplo de uma classe criada em Java com seus atributos.**



Fonte: Autoria própria, 2019.

Dito por Furgeri (2015), pode se criar objetos em uma classe e esses objetos podem ser utilizados em outras classes ou aplicações. Para utilizar um objeto, terá que seguir os seguintes passos:

* Declaração de objetos: Segue um padrão de nome-do-tipo nome-da-variável. Pra declaração de um objeto usa a seguinte sintaxe: nome-da-classe nome-do-objeto. Para gerar um objeto um objeto de classe por exemplo, Carro, sua sintaxe será: Carro carro1. O nome Carro se refere à classe em si, e carro1 trata-se de uma declaração dizendo que um objeto do tipo carro. Se criar vários objetos de uma classe, como no exemplo “Carro”, se pode ter os objetos “carro1”, “carro2”, “carro3” e assim por diante;
* Instanciação do objeto: É a criação pela sua alocação de memória para armazenar informações sobre ele, que se reserva endereços da memória para armazenar dados correspondentes. Para que se possa fazer a instanciação de um objeto, será usado o operador “new”. Por exemplo Carro carro1 = new Carro().

**Figura 8: instancia de um objeto**



Fonte: Autoria própria, 2019.

## UML

A UML é uma linguagem visual para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados na modelagem de sistemas. Esses elementos permitem representar os conceitos do paradigma da orientação a objetos. Através dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que representam diversas perspectivas de um sistema. (Bezerra, 2007, p15).

Ainda, Guedes (2011) porem muitos se confundem achando que UML é uma linguagem de programação, mas não é. O objetivo dela é auxiliar os engenheiros de softwares a definirem as características do sistema a ser feito pela empresa, tais como seus requisitos, seu comportamento, estrutura lógica, dinâmica de processos e necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado.

Guedes (2011, p19), “A UML surgiu da união de três métodos de modelagem: método Booch, método OMT (Object Modeling Technique) de Jacobson e o método OOSE (Object-Oriented Software Engineering) de Rumbaugh, no início da década de 1990.”

Segundo Góes (2014), a UML pode ser utilizada como base as análises, projeto e implementação, pois nos permite pensar antes de codificar, utilizando um conjunto de diagramas que permitem uma comunicação entre as equipes de desenvolvimento.

### Levantamento de requisitos

Conforme Guedes (2011), uma das primeiras etapas desse processo denomina-se levantamento e análise de requisitos. O método de desenvolvimento de Software está dividido em quatro fases, sendo elas: O levantamento de requisitos; logo depois a análise dos requisitos e feito o projeto do software; A construção do software e os testes deles; E por fim a implantação dele.

De acordo com Bezerra (2007), na etapa de levantamento e análise de requisitos, procura-se compreender os problemas dos clientes e assim achar uma solução para desenvolver um software que foi solicitado pelo cliente. A forma de descobrir seria por meio de entrevistas com usuários que irão usar o futuro software.

Segundo Guedes (2011), fala da existência de dois tipos de requisitos, que são os requisitos funcionais e não-funcionais. Os requisitos funcionais seriam às funcionalidades do *software*, ou seja, aquilo que o cliente deseja que o software faça. Já os requisitos não-funcionais determinam como serão aplicados os requisitos funcionais, que correspondem as restrições, validações, condições que os requisitos funcionais possam realizar.

Ainda, Bezerra (2007), na fase de construção e de testes, o sistema é codificado, ou seja, ocorre a tradução do código para a execução do software. E pode ser em uma ou mais linguagens de programação. Já na fase de testes, são realizados diversos teste do sistema em construção e o principal trabalho é encontrar os erros para poder assim solucioná-los.

Segundo Góes (2014), depois das análises e levantamento dos requisitos temos a implantação do sistema. Onde se é entregue o software pronto para uso do usuário, os manuais, os dados importados para o sistema e os treinamentos para os usuários.

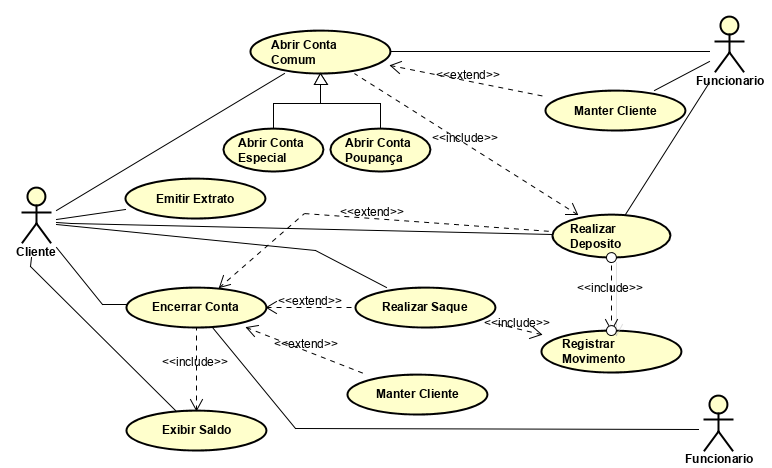
### Diagramas de Caso de Uso

O objetivo disto é fornecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos, procurando-se, assim, atingir a completitude da modelagem, permitindo que cada diagrama complemente os outros. (Guedes, 2011, p19).

Existem diversos diagramas de casos de Uso, porem aqui será abordado somente os que serão usados nesse projeto. Tal como, Diagrama de Atividade, Diagrama de classes e Diagrama de sequência.

Como diz Guedes (2011), o diagrama de caso de uso é usado nas fases de levantamento e análise de requisitos, embora possa ser usado no restante das fases de desenvolvimento do software. Onde se identificasse os autores (usuários e sistemas) e as funcionalidades que o sistema realiza-la. Um diagrama que serve de base para outros diagramas e é comumente consultado durante todo o processo de modelagem. Veja o exemplo na figura abaixo:

**Figura 9: Exemplo de diagrama de caso de uso**

****

Fonte: UML 2 uma abordagem prática, 2011.

Segundo Góes (2014), a principal função deste diagrama é apresentar um sistema que mostre em uma visão mais clara as informações para os usuários. Mostrando os módulos que compõem o sistema, os usuários e os papéis de cada um que iram desempenhar para seu funcionamento.

Ainda Bezerra (2007), diz que existem os relacionamentos de inclusão e exclusão. O includeé um relacionamento onde ocorre uma obrigatoriedade entre casos de uso. É representado por uma reta tracejada com uma seta que aponta para o caso de uso incluído. Nota-se que o sistema faz com que o ator tenha que efetuar o login, para usar o sistema. O *include*, portanto, ocorre quando uma situação é dependente da outra.

### Diagrama de atividades

Conforme Guedes (2011), o diagrama de atividades procura descrever os passos do usuário até a atividade que ele deseja fazer. Antigamente era conhecido como diagrama de gráfico de estados. A partir da UML 2.0, foi considerado independente do diagrama de máquina de estados.

Na figura a seguir, podemos ver um diagrama de atividade:

**Figura 10: Exemplo de diagrama de atividade**

****

Fonte: Aprenda UML por meio de Estudos de Caso (janeiro/2014).

Tem início a partir do círculo preto, sendo este o nó inicial. Em seguida, a figura retangular é uma ação, que neste caso seria uma impressão de requisição. Logo em seguida vem ações que devem ser subsequentes a anterior, como: separar produtos, embalar produtos e emitir nota fiscal. E por fim chegando ao círculo preto, onde este seria o nó final. Essas ações são representações dos atos dos usuários e das reações do sistema. Ligando essas ações, existe um segmento de reta com uma seta que aponta para a próxima ação. Esses segmentos são os fluxos de controle, que servem para conectar as ações.

Segundo Góes (2014), O diagrama acima representa os aspectos dinâmicos e pode ser utilizado para modelar um sistema de informação, podendo assim ser módulos dos sistemas, código dos programas ou os processos (fluxos de trabalho) de uma organização.

Conforme Bezerra (2007), o diagrama de atividade pode-se dividir em 2 etapas, os fluxos de controle sequencias e os de controle paralelos.

A modelagem de atividade enfatiza a sequência e condições para coordenar comportamentos de baixo nível. Dessa forma o diagrama de atividade é o diagrama com maior ênfase ao nível de algoritmo da UML e provavelmente um dos mais detalhistas. (Guedes, 2011, p. 277).

### Diagrama de classes

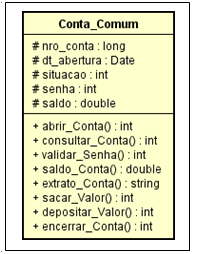
Bezerra (2007, p. 112) “O diagrama de classes é utilizado na construção do modelo de classes desde o nível de analise até o nível de especificação. ”

Segundo Guedes (2011), o principal trabalho dele é visualizar as classes que farão parte do sistema. Os atributos e os métodos. Recomenda-se que utilize o diagrama de classe na fase de analise, pois pode ir construindo um modelo conceitual das informações necessárias do software.

Como diz Góes (2014), a ideia central desse diagrama é concentrar a construção do sistema, ou seja, deixar ela mais próxima do real possível para contrução do software.

A figura abaixo é um exemplo de classe, que é dividida em três partes na seguinte ordem: nome da classe, atributos e métodos.

**Figura 11: diagrama de Classe**



Fonte: UML 2 Uma abordagem pratica, 2011, p. 102.

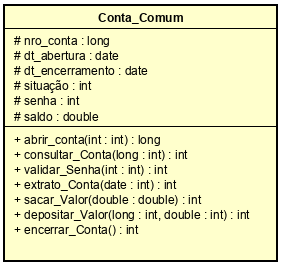
Na segunda parte é onde fica os atributos da classe Conta\_Comum e seus tipos de dados (dados que servem para armazenar algo do atributo). Como por exemplo, na figura acima, atributo nro\_conta é do tipo *long*, já a dt\_abertura é do tipo *date.*

Na terceira parte do diagrama de classes, podemos ver os métodos.

Guedes (2011, p. 103) “Os símbolos de sustenido (#) e mais (+) na frente dos atributos e dos métodos representam a visibilidade dos mesmos, o que determina que objetos de quais classes podem utilizar o atributo ou o método em questão.”

Continuando Guedes (2011), os métodos podem receber valores como parâmetros e retornar valores, igual a linguagem C. Que pode ser um resultado produzido por uma execução ou um valor representado para informar que se obteve sucesso ou não. Por exemplo, na classe abrir\_Conta é um long, que será um número de uma nova conta gerada pelo sistema, quanto no validar\_Senha é um número inteiro (int) utiliza-se para descobrir se o método foi realizado com sucesso. Alguns métodos podem ter vários parâmetros, como pode se ver na figura a seguir:

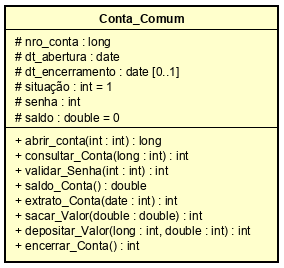
**Figura 12: Detalhamento das assinaturas das operações**

****

Fonte: UML 2 Uma abordagem pratica, 2011, p. 104.

Aqui podemos ver que alguns métodos receberam alguns parâmetros diferentes um dos outros, por exemplo, abrir\_Conta recebeu um int(inteiro), consultar\_Contarecebeu um long, enquanto que depositar\_Valor recebeu um long e um double.

**Figura 13: detalhamento dos Atributos**

****

Fonte: UML 2 Uma abordagem pratica, 2009, p. 111.

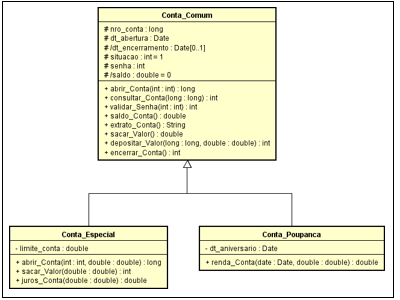
Segundo Guedes (2011), aqui nesse exemplo podemos ver que nos atributos situação e saldo será dado 1 e 0 respectivamente quando houver uma abertura de conta, o que significa que está ativa. Já no atributo dt\_encerramente recebeu os valores [0...1], chama-se multiplicidade, o que significa que existira no mínimo nenhuma (0) e no máximo (1) data de encerramento para a conta.

Segundo Bezerra (2007), existem também o relacionamento de generalização consiste em permitir que um ator herde características de outro ator (base). O herdeiro poderá especializar algum comportamento do autor base. A vantagem da herança é que só precisa mexer no comportamento do “ator base”, pois mudará com os herdeiros também. As classes mais específicas são chamadas de subclasses ou classes filhas. Os objetos dessas subclasses herdam atributos e operações das superclasses.

No pensamento de Guedes (2011), o objetivo dessa associação é representar a ocorrência de herança entre as classes-mãe e as classes filhas. Além disso métodos podem ser redeclarados de uma classe especial, com mesmo nome porém com um comportamento diferente sem mudar o código-fonte, pois o nome do método não foi alterado, somente foi redeclarado e se comporta de maneira diferente quando for chamado.

A seguir podemos ver um exemplo de relacionamento de generalização.

**Figura 14: Generalização ou Especialização**



Fonte: UML 2 Uma abordagem pratica, 2009, p. 114

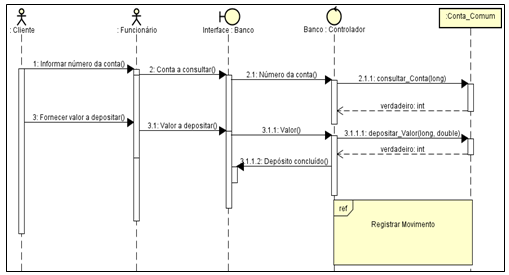
### Diagrama de sequência

Segundo Góes (2014), os diagramas de sequência descrevem numa linha de tempo a sequência das comunicações entre objetos de um sistema de informação. Seus principais objetivos são: documentar casos de uso, mostrar a comunicação do sistema por meio de mensagens em ordem de tempo, e assim validar todas as operações das classes foram identificadas e declaradas ou ainda validar a existência de um objeto necessário ao funcionamento do Sistema.

Segundo Guedes (2011), o diagrama de sequência é baseado no diagrama de casos de uso, sendo assim, todo diagrama de caso de uso precisa ter um diagrama de sequência. Ele também é uma excelente forma de validar e documentar o diagrama de classes, porque consegue-se ver os métodos necessários para declarar as classes. O diagrama de sequência é composto por:

* Atores: São instancias dos atores declarados no diagrama de caso de uso.
* Lifelines: É um participante individual emu na interação que irá se referir a uma instância de classe.
* Linha de vida: Representa o tempo em que um objeto lifeline existe durante um processo e são representadas por linhas travejadas. Quando o objeto é destruído a linha é interrompida por um “X”.
* Foco de controle ou ativação: Determina os períodos que um objeto está participando e identifica os momentos que um objeto está executando um ou mais métodos.
* Mensagens ou estímulos: Utiliza-se para ocorrência de eventos, que faz a chamada de um método de algum objeto ou pode também ocorrer uma mensagem entre dois atores. As mensagens podem ser de atores e outro ator; ator e um objeto; objeto e outro objeto; objeto e um ator.
* Mensagem de retorno: São representadas por uma linha tracejada contend uma seta que aponta para o objeto que recebe o resultado do método. São as respostas das mensagens dos objetos ou atores que a chamou. Onde se torna informações especificas do método ou um valor do método que foi executado.
* Auto chamadas ou Auto delegação: São mensagens que um objeto envia para si mesmo.
* Detalhes de tempo: Onde se define o tempo de uma mensagem. Como por exemplo pode ser colocar um tempo máximo de espera. Como padrão é apresentada na diagonal.
* Portas: Representa no objeto instância das portas declaradas na classe que ele pertence, obtendo assim ais de uma linha de vida para que ocorra mensagens externas e internas no objeto.

**Figura 15: Diagrama de sequência**



Fonte: UML 2 Uma abordagem pratica, 2011, p. 206.

Conforme Bezerra (2007), esse diagrama consiste nas trocas de mensagens entre os objetos na ordem temporal em que as ações irão acontecendo.

## NoSql

NoSql foi criado para facilitar a distribuição. Isso implica um diferente modelo de consistência do banco de dados relacionamento, segundo Cardoso (2012).

Diz Lobo, Almeida, Lobato (2018), em 1998 surgiu o termo NoSql, como uma solução de banco de dados que não oferecia interface SQL, mas com sistemas baseados na arquitetura relacional. O principal propósito do NoSql não é substituir o modelo relacional, como um todo e sim apenas em casos que seja necessária uma maior flexibilidade de estruturação do banco.

Conforme Oliveira (2014), Devido ao crescimento da internet o banco de dados relacional mostrou não recomendável para aplicações que geram grande volume de memória, com isso foi criado o banco de dados não relacional, que se adequa melhor a aplicação que utilizam grande volume de memória.

Segundo Paniz (2016), existe vários modelos de banco de dados não relacional e eles são classificados como seu tipo de armazenamento, segue tipos:

### Chave/Valor

Como diz Oliveira (2014), é uma aplicação de matrizes ou algoritmos para efetuar uma busca em todos registros dos arquivos compartilhados. Comum ser usado em programas de compartilhamento e de mudanças constantes.

Segundo Cardoso (2014), esse é o modelo mais importante no mundo NoSql já que todos outros tipos derivam dele. Ele tem um conjunto de pares que seria a chave/valor, ter uma chave e logo em seguida o valor.

Dito por Paniz (2016), ele é extremamente simples, em comparação com banco de dados relacional, seria um bando de com uma tabela e duas colunas, sendo que uma das colunas é a chave primária e a outra o valor.

**Figura 16: Chave/Valor**



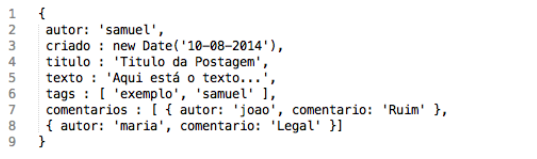
Fonte: Cardoso, 2014.

### Orientado a objetos

É baseado no armazenamento de pares de chave/valor, com um esquema altamente flexível. Com essa característica torna ele uma ótima opção para dados semiestruturados, como utilizados em ferramentas web colaborativas, segundo Oliveira (2014).

Conforme Cardoso (2012), é um modelo do NoSql na qual cada entrada da sua basse corresponde a um documento. Na qual estes documentos são geralmente decompostos num identificador e num valor. Nesses valores tem a possibilidade de colocar mais identificadores.

**Figura 17: Exemplo de orientado a objetos**



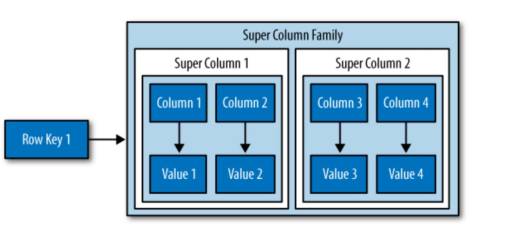
Fonte: Oliveira, 2014.

### Orientado a colunas

É um modelo mais complexo que chave/valor, em que se muda o paradigma da orientação à registro com a orientação á colunas. Com isso nem todas as linhas terão a mesma quantidade de colunas, segundo Oliveira, 2014.

Dito por Cardoso (2012), é uma evolução da modelo chave/valor. Chegou a conclusão que pares de chaves sem o mínimo de critério de organização seria um incomodo. Então para agrupar de forma organizada os pares de chave/valor poderiam ser agrupados em linhas. Dessa forma surgiu o conceito superfamília de colunas, que seria um para chave/valor constituído por um id e um conjunto de colunas.

**Figura 18: Exemplo de orientado a colunas.**



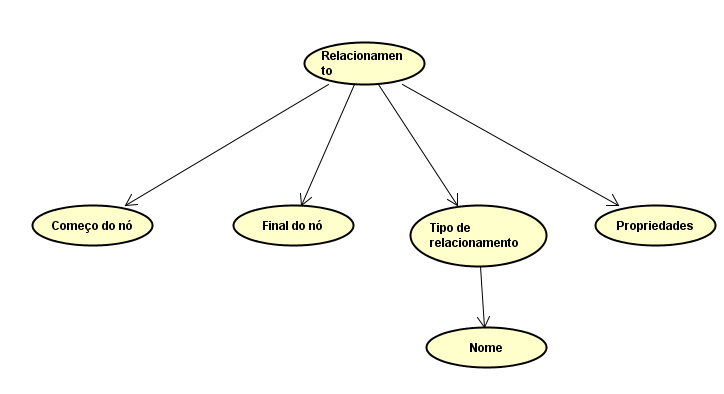
Fonte: Cardoso, 2012.

### Banco de dados de grafos

Tipo de banco de dados mais complexos, nesse banco de dados se guarda objetos e não registros como nos demais tipos. Para efetuar a pesquisas desses itens é feita navegação desses objetos. Banco de dados nessa classificação armazena vértices e arestas, segundo Oliveira, 2014.

Segundo Cardoso (2012), esse modelo tem um desempenho maior em comparação ao banco de dados relacional, em consultas mais complexas que se utiliza joins, estrutura de grafos simplificam muito estas queries mais complexas, que chega a ser melhor que o banco de dados relacional.

**Figura 19: Exemplo de Grafos**



Fonte: Autoria própria, 2019

# NBR 5410

NBR (2004), esta norma estabelece as condições a que devem satisfazer as instalações elétricas de baixa tensão, a fim de garanti a segurança de pessoas e animais, o funcionamento adequado da instalação e a conservação de bens.

Esta norma aplica-se principalmente às instalações e edificações, qualquer que seja seu uso (residencial, comercial, público, industrial, de serviços, agropecuário, hortigranjeto, etc.), incluindo as pré-fabricas:

1. Esta norma aplica-se também as instalações elétricas;
2. Em áreas descobertas das propriedades, externas as edificações;
3. De reboques de acampamentos (trailers), locais de acampamentos(campings), marinas e instalações analógicas;
4. Canteiros de obra, feiras, exposições e outras instalações temporárias.

Esta norma aplica-se:

1. Aos circuitos elétricos alimentados sob tensão nominal igual ou inferior a 1.000 V em corrente alternada com frequência a 400 HZ, ou a 1.500 HZ em corrente contínua;
2. Aos circuitos elétricos, que não os internos aos equipamentos, funcionando sob uma tensão igual ou inferior a 1.000 V em corrente alternada (por exemplo, circuitos de lâmpadas a descarga, precipitadores eletrostáticos etc.);
3. A toda fiação e a toda linha elétrica que não sejam cobertas pelas normas relativas aos equipamentos de utilização;
4. Linhas elétricas fixas de sinal (com execução dos circuitos internos dos equipamentos).

Esta norma aplica-se as instalações novas e a reformas em instalações existentes.

Esta norma não se aplica a:

1. Instalações de tração elétrica;
2. Instalações elétricas de veículos automotores;
3. Instalações elétricas de embarcações e aeronave;
4. Equipamentos para supressão de perturbação radioelétrica, na medida que não comprometam segurança das instalações;
5. Instalações de iluminação pública de distribuição de energia elétrica;
6. Redes públicas de distribuição de energia elétrica;
7. Instalações de proteção contra quedas diretas de raios. No entanto, esta norma considera as consequências dos fenômenos atmosféricos sobre as instalações (por exemplo, seleção dos dispositivos de proteção contra sobre tensão);
8. Instalações em minas;
9. Instalações de cercas eletrificadas.

Os componentes de instalações são considerados apenas no que concerne à sua seleção e condições de instalação. Isto é igualmente válido para conjuntos em conformidade com as normas a elas aplicáveis.

A aplicação desta norma não dispensa o atendimento a outras normas complementares, aplicáveis a instalação e locais específico.

A aplicação desta norma não dispensa o respeito aos regulamentos de órgão público aos quais a instalação deva satisfazer.

As instalações elétricas cobertas por esta norma estão sujeitas também, naquilo que for pertinente, às normas para fornecimento de energia estabelecidas pelas autoridades reguladoras e pelas empresas distribuidoras de eletricidade.

### Iluminação

NBR (004), em cada cômodo ou dependência deve ser previsto pelo menos um ponto de luz no teto, comandado por interruptor.

Na determinação das cargas de iluminação, como alternativa a aplicação da ABNT NBR 5413 (Conforme prescrito na alínea a) de 4.2.1.2.2, pode ser adotado os seguintes critérios:

1. Em cômodos ou dependências com áreas superiores a 6 m², deve ser prevista uma carga mínima de 100 VA;
2. Em cômodos ou dependências com área superior a 6 m², acrescida 60 VA para cada aumentos de 4 m² inteiros.

### Pontos de tomadas

NBR (2004), os números de pontos de tomadas devem ser determinados em função de destinação do local e dos equipamentos elétricos que podem ser aí utilizados, observando-se no mínimo os seguintes critérios:

1. Em banheiros, deve ser previsto pelo menos um ponto de tomada, próximo ao lavatório;
2. Em cozinhas, copas, copas-cozinhas, área de serviço, cozinha-área de serviço, lavandeiras e locais análogos, deve ser previsto no mínimo um ponto de tomada a cada 3,5 m, ou, fração, de perímetro sendo que acima de bancada da pia devem ser prevista no mínimo duas tomadas de corrente, no mesmo ponto ou em pontos distintos;
3. Em varandas, deve ser previsto pelo menos um ponto de tomada;
4. Em salas e dormitórios devem ser previstos pelo menos um ponto de tomada para cada 5 m, ou fração de perímetro, devendo esses pontos ser espaçado tão uniformemente quando possível.

Em cada um dos cômodos e dependências de habitação devem ser previstos pelo menos:

1. Um ponto de tomada, se a área do cômodo ou dependência for igual ou inferior a 2,25 m². Admite-se que esse ponto seja posicionado externamente ao cômodo ou dependência, até 0.80 m no máximo de sua porta de acesso;
2. Um ponto de tomada, se a área do cômodo ou dependência for superior a 2,25 m² e igual ou inferior a 6m²;
3. Um ponto de tomada para cada 5 m, ou fração, de perímetro, se a área do cômodo ou dependência for superior a 6 m², devendo esses pontos ser espaçados tão uniformemente quanto possível.

A potência a ser atribuída a cada ponto de tomada é função dos equipamentos que ele poderá vir alimentar e não deve ser inferior aos seguintes valores mínimos:

1. Em banheiros, cozinhas, copas, copas-cozinhas, áreas de serviço, lavandeiras e locais análogos, no mínimo 600 VA por ponto de tomada, até três pontos, e 100 VA por ponto para os excedentes, considerando-se cada um desses ambientes separadamente. Quando o total de tomadas no conjunto desses ambientes for superior a seis pontos, admite-se que critério de atribuição de potência seja de no mínimo 600 VA por ponto de tomada, até dois pontos, e 100 VA por ponto para os excedentes, sempre considerado cada um dos ambientes separadamente, nos demais cômodos ou dependências, no mínimo 100 VA por ponto de tomada.

# Desenvolvimento

# Conclusão

# Referências

FURGERI, Sérgio. **Java 8 - Ensino Didático:** Desenvolvimento e Implementação de Aplicação. 2. ed. São Paulo: Érica Ltda, 2015. 320 p.

BARREIRO, Daniela; SOBRAL, João Bosco Mangueira. **Programação em Java.** Florianópolis: Copyleft Pearson Education, 2008. 89 p.

DEITEL, Paul; DEITEL, Harvey. **Java:** Como Programar. 10. ed. São Paulo: Pearson Education do Brasil Ltda, 2016. 934 p.

FARIA, Thiago; NORMANDES JUNIOR, **Java e Orientação a Objetos.** Uberlândia: Algaworks Softwares, Treinamentos e Serviços Ltda, 2014. 272 p.

CARDOSO, Ricardo Manuel Fonseca. **Bases de Dados NoSQL.** 2012. 81 f. Dissertação (Mestrado) - Curso de Engenharia Informática, Área de Especialização em Arquiteturas de Sistemas de Redes, Instituto Superior de Engenharia do Porto, Porto, 2012.

LOBATO, Antonio Soares; LOBO FILHO, Jorge Rogério Pinheiro; OLIVEIRA, Thiago Afonso. NoSQL: Ferramentas, desafios de aplicabilidade em um case real. **Revista de Tecnologia da Informação e Comunicação da Faculdade Estácio do Pará**, Belém, v. 1, n. 1, p.69-74, 06 abr. 2018.

OLIVEIRA, Samuel Silva de. Banco de dados não relacionais: Um novo paradigma para armazenamento de dados em sistemas de ensino colaborativo. **Revista da Escola de Administração Pública do Amapá**, Amapá, v. 1, n. 2, p.184-194, 01 ago. 2014.

PANIZ, David. **NoSQL:** Como armazenar os dados de uma aplicação moderna. São Paulo: Casa do Codigo, 2016. 177 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **5410**: Instalações elétricas de baixa tensão. 1 ed. Rio de Janeiro: Abnt, 2004. 209 p.