

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS  
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO.  
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS.



PROJETO ARQUITETURAL E MODELAGEM CONCEITUAL

BRUNO CAMARGO MANSO  
JOÃO VICTOR CARDOSO

GOIÂNIA, GO  
2021

BRUNO CAMARGO MANSO  
JOÃO VICTOR CARDOSO

PROJETO ARQUITETURAL E MODELAGEM CONCEITUAL

Matéria: Desenho de Aplicativos para Dispositivos Móveis  
Orientador: Fabrício Schlag

GOIÂNIA, GO  
2021

<b>Introdução</b>	<b>3</b>
<b>Elaborando o Diagrama de Classe</b>	<b>3</b>
<b>Melhoria utilizando Classes Abstratas</b>	<b>4</b>
<b>Melhoria utilizando Prototypes</b>	<b>5</b>
<b>Conclusão</b>	<b>6</b>

## Introdução

O presente trabalho tem como objetivo elaborar um diagrama de classes segundo os requisitos dados e cujo tema é um Sistema Acadêmico. Iremos também, propor um conceito arquitetural de modo que tal diagrama seja otimizado seguindo os conceitos de melhoramento contínuo. Serão, então, diferentes abordagens para o mesmo problema apresentado.

Desta forma, teremos a oportunidade de aplicar em uma situação simulada, arquiteturas diferentes, para a melhor fixação do conhecimento, os alunos se comprometem a elaborar tais diagramas de forma crítica, garantindo assim um contínuo melhoramento na cadeia de processos.

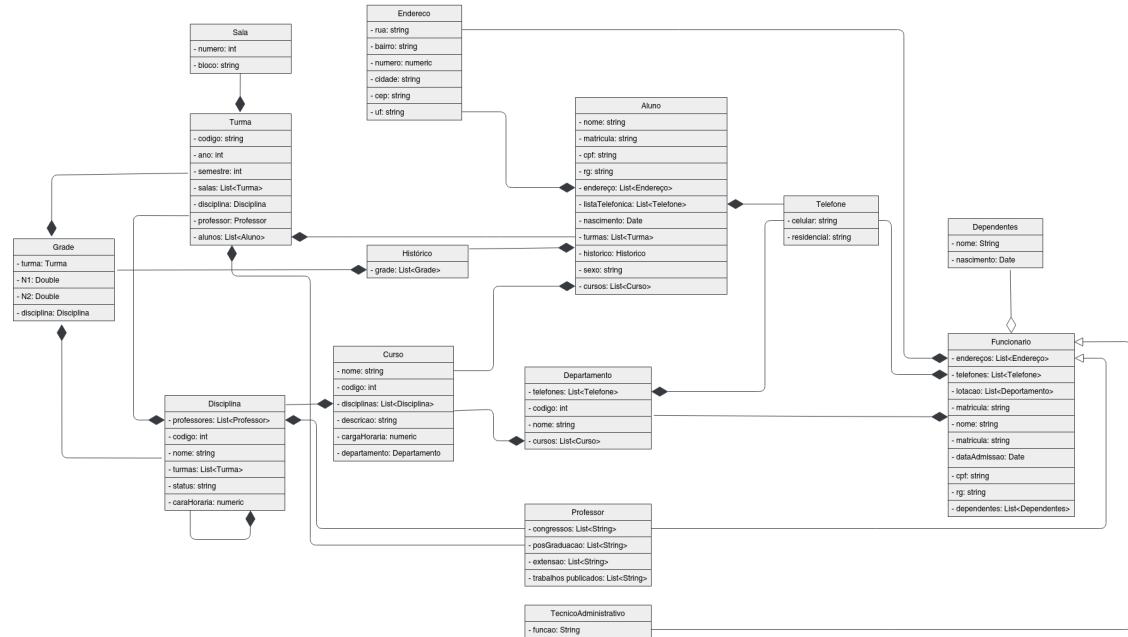
## Elaborando o Diagrama de Classe

A elaboração dos diagramas a seguir demandaram tempo e acurácia. Os requisitos descritos continham altos níveis de abstração, e deveriam então serem simbolizados de forma clara em um diagrama de classe. Utilizamos da técnica ‘dividir para conquistar’ no sentido de fragmentar de forma positiva os relacionamentos entre classes.

Boa parte de nossos esforços foram no sentido de reduzir as redundâncias, mantendo sempre a coesão e, conforme nosso trabalho tomava novas arquiteturas, no sentido de melhorar a coesão e o aproveitamento de código, com menor acoplamento entre classes.

Nossa primeira abordagem fora utilizar da Orientação a Objetos, isso otimizaria drasticamente o reaproveitamento de classes, facilitando o uso de de sobrescrita e sobrecarga de métodos bem como a criação de vários objetos abstratos correspondentes às principais abstrações dadas pelos requisitos.

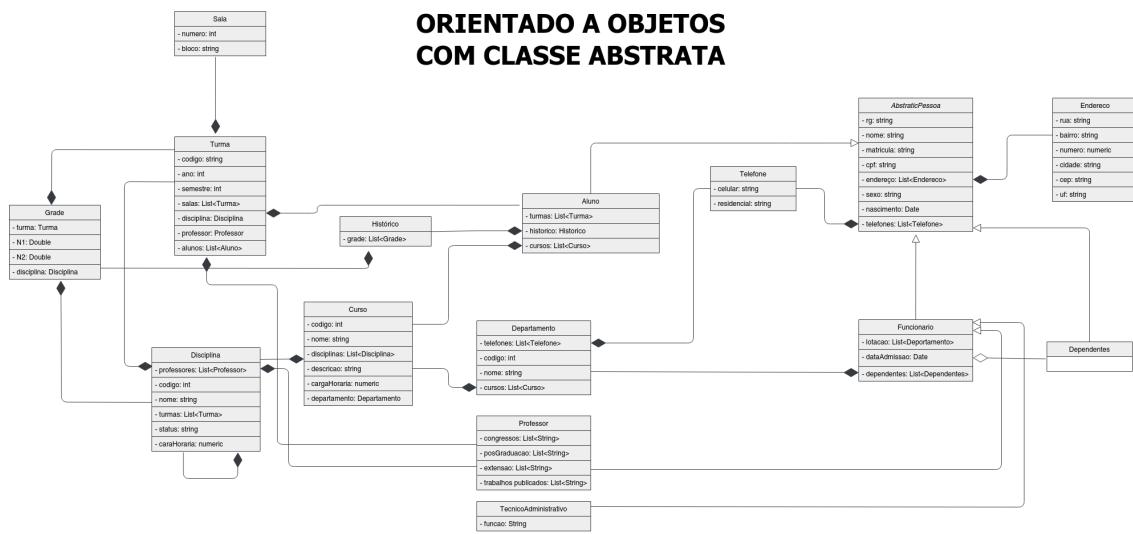
## ORIENTADO A OBJETOS



A figura acima mostra o uso massivo de objetos de forma que cada um carrega seus devidos atributos necessários. Isso obedece às regras de programação orientada a objetos, fazendo com que cada classe tenha acesso, de alguma maneira, a outra classe. Pensar na coesão foi o maior desafio nesse momento, mas os conceitos sobre entidade e relacionamento ajudaram a elucidar como tal ligação deveria ocorrer.

## Melhoria utilizando Classes Abstratas

Agora é o momento de aplicarmos um novo conceito que deixaria nosso código mais enxuto, aproveitando os recursos de Classes Abstratas oferecidas pelo Java.



Na figura acima percebe-se que uma nova classe será criada. Mais uma abstração que irá além das abstrações propostas nos requisitos dados. A classe Pessoa servirá para aluno bem como servirá a funcionário, que por sua vez servirá a professores. As classes 'Funcionário' e 'Aluno' seriam classes filhas, enquanto a classe 'Abstract Pessoa' seria a classe pai. Criamos então um sistema hierárquico que cede atributos às classes mais baixas.

Ou seja, as filhas herdariam os atributos da sua classe antecessora. Isso reduz drasticamente o acoplamento de código, melhorando a coesão e a velocidade de processamento em ações de buscas e *Crud* e até mesmo a diminuição da quantidade de dados no diagrama, tornando-o também mais sucinto.

## Melhoria utilizando Prototypes

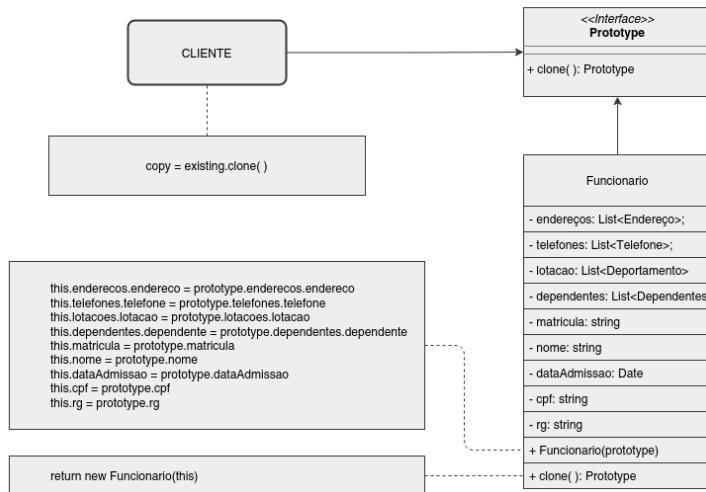
Em nossa nova melhoria, resolvemos apanhar um trecho do diagrama para exemplificá-lo, assim damos maior ênfase de como seria uma ação de um cliente/usuário e como a arquitetura voltada a protótipos poderia resolver alguns problemas.

Usamos então um padrão criacional que permite a geração de novos objetos a partir de um modelo original, ou seja, os objetos são clonados. Em vez de o cliente implementar um código que utiliza o operador 'New', este utiliza o método *Clone()* presente no protótipo.

O padrão 'New' fora então abandonado dando lugar ao clone do protótipo que pode também ser associado ao método *Abstract Factory* que o ajudaria a gerar vários clones se necessário.

Então, o prototype é uma classe com o objetivo de declarar uma interface para objetos capazes de clonar a si mesmo. Veja a seguir:

# ORIENTADO A OBJETOS COM PROTOTYPE



## Conclusão

Criar um diagrama de classes completo, já com todas as suas relações, pode ser uma tarefa mais simples do que sua utilização. A criação de códigos por meio de padrões de projeto e boas práticas, pode ser uma tarefa que leva tempo para aprender, e até mesmo saber o momento certo de usar.

Usamos boas práticas presentes em diversas linguagens. Uma delas foi a implementação de classes abstratas, que utilizam de um sistema de heranças. Também fizemos bom uso de um padrão de projeto criacional - *Prototype* - que otimizou a utilização de alguns objetos sem a necessidade de recriá-los.

Levando em conta o número de padrões de projetos existentes, utilizar apenas um, pode parecer pouco, mas o fato de um diagrama de classes ser grande e complexo, não significa que o mesmo necessite de diversos padrões para ser elaborado/otimizado, todavia, como um diagrama menor, também não significa que o mesmo terá menos padrões implementados.

Cada caso é um caso. A aplicação de um padrão requer estudo e conhecimento, pois, mesmo que seja um padrão utilizado diversas vezes e que acaba se tornado um padrão, pode ser que a aplicação do mesmo, naquele momento, não seja o ideal. É sobre isso que tratamos no presente trabalho. Boas práticas, seguindo padrões arquiteturais, podem, juntos, melhorar a otimização em termos de velocidade, coesão, acoplamento e consequentemente, no entendimento do diagrama/código e nas manutenções futuras.