Trabalho de POO:
(To Do List)

GRUPO DO TRABALHO:

Guilherme Santos, Igor Braziel e Pablo Oliveira

Padrões de projeto utilizados:

- Padrão Criacional: **Singleton** (Apresentado pelo professor)
- Padrão Estrutural: **Decorator** (Padrão de projeto escolhido para apresentar no seminário)
- Padrão comportamental: **Observer** (Nossa escolha)

Entidades importantes:

- **ToDoList (Classe):**

  Nossa lista de tarefas (to do list) será instância da classe ToDoList e deverá ser a única instância, por isso
  essa classe utilizará do padrão de projeto Singleton, essa lista será composta por tarefas, que serão instâncias
  da classe Task

- **Task (Classe):**

  as tarefas terão os seguintes atributos:
  (id, nome, nível de dificuldade, prazo final, status de progresso, porcentagem feita), também foi implementado o padrão de projeto Observer, a classe ToDoList implementará
  a interface Observable e a classe Task implementará a interface Observer, quando ocorrer alguma
  alteração na ToDoList, todas as instâncias de Task serão notificadas e devidamente atualizadas conforme a necessidade

- **ListDecorator (Classe abstrata):**

  A classe abstrata **ListDecorator** conterá um atributo (list) do tipo IList, referente ao objeto que vai ser decorado, a classe simplesmente chama todos os métodos e passa a responsabilidade deles para essa instância, essa classe serve de base para os *Decorators* concretos.

- **IList (interface):**

  A interface IList contém todos os métodos que devem ser implementados pelas classes referentes ao padrão de projeto Decorator.

Onde foi utilizado o padrão Singleton:

- A classe ToDoList irá utilizar do padrão de projeto Singleton, a classe possui um atributo privado e estático (de classe), chamado ***instance*** do tipo ToDoList, ele conterá a única instância de classe que deverá ser criado, o construtor da classe é privado, para obter acesso à essa instância, é necessário invocar o método estático getInstance(), utilizando ToDoList.getInstance(), ao ser chamado pela primeira vez, ele irá criar a instância da classe, nas chamadas futuras do método, ele simplesmente retornará a referência para essa instância.

Onde foi utilizado o padrão Observer:

- a classe ToDoList implementará a interface Observable, que possui a declaração dos métodos ***notifyObserver()*** e ***notifyObservers()*** e a classe Task implementará a interface Observer, que possui a declaração do método ***update()***, quando ocorrer alguma alteração na ToDoList, todas as instâncias de Task serão notificadas através destes métodos e chamar o método ***update()*** nos objetos necessários.

Onde foi utilizado o padrão Decorator:

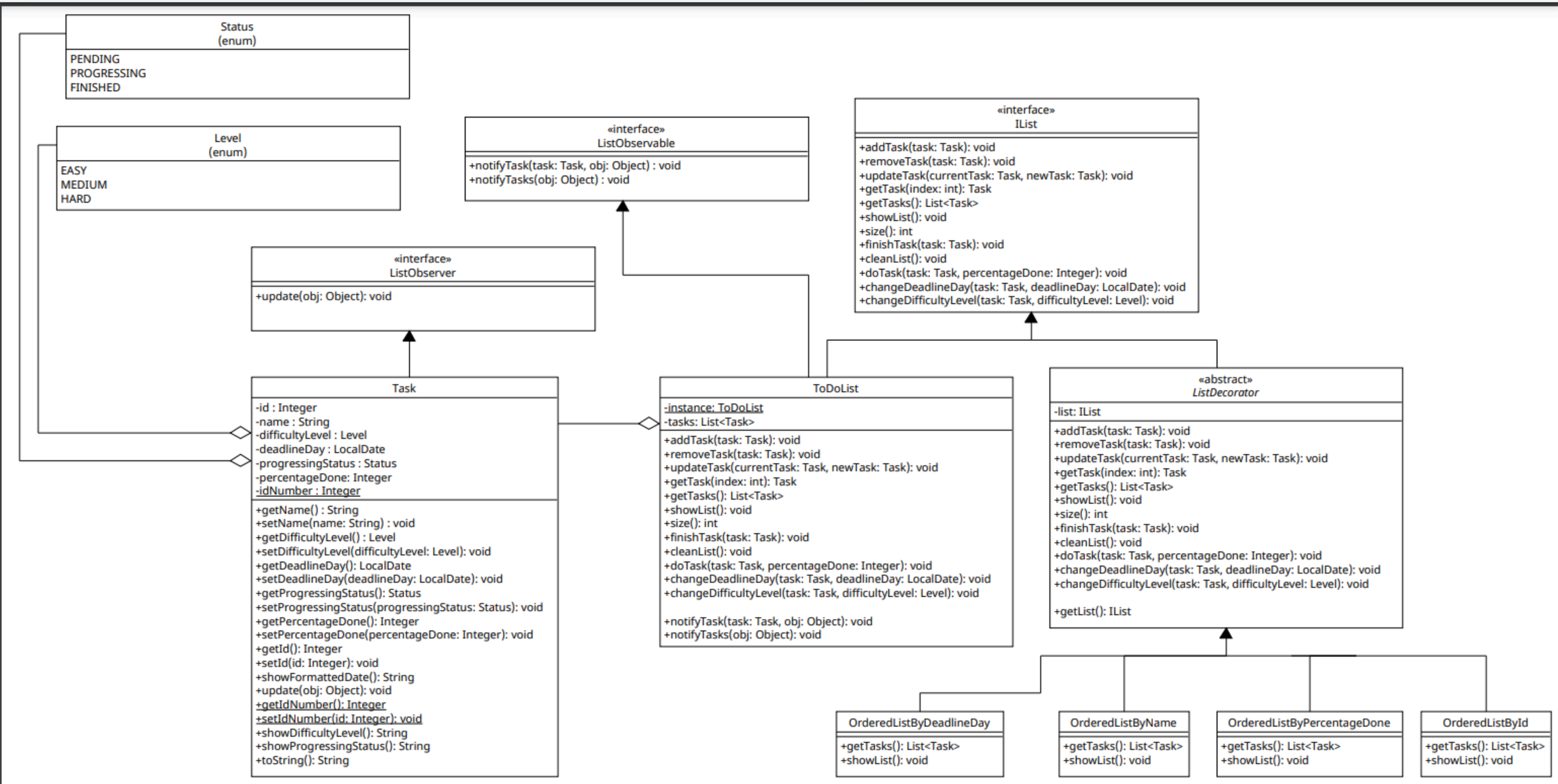
- as classes ToDoList, ListDecorator deverão implementar a interface IList, que contém todos os métodos que deverão ser implementados, a classe ListDecorator é abstrata e serve de base para as classes que herdam dela, que são (OrderedListByDeadlineDay, OrderedListById, OrderedListByName, OrderedListByPercentageDone) essas classes são concretas e adicionarão funcionalidades à lista de tarefas em tempo de execução, caso seja solicitado pelo usuário.

Interface gráfica:

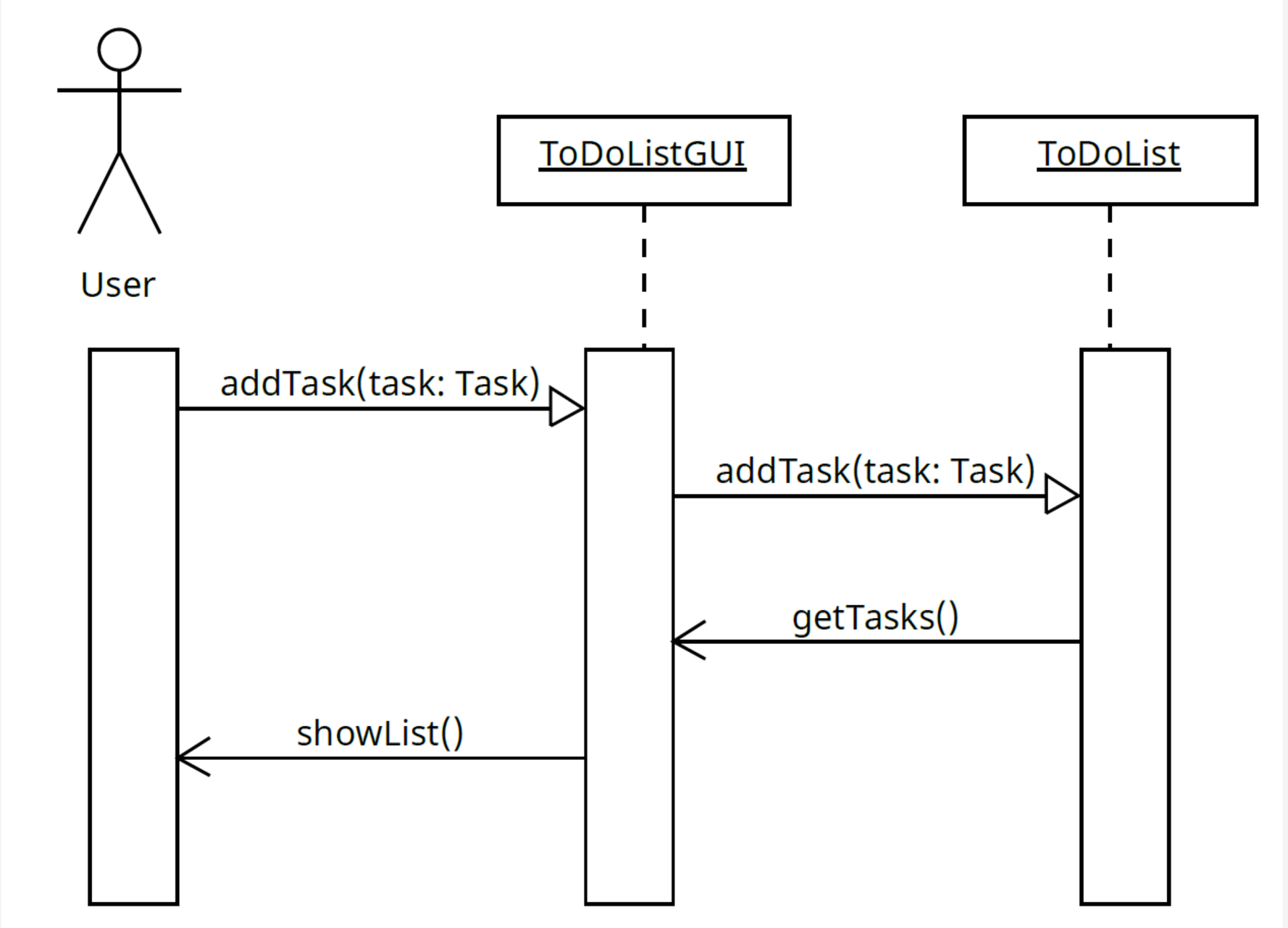
- Foi criado uma interface gráfica para a interação do usuário com o programa, essa interface é bastante simples mas ilustra todas as funcionalidades do programa, a interface vai exibir a lista de tarefas e através dos botões o usuário pode interagir com essa lista

- As funcionalidades possíveis são: Adicionar tarefas, remover tarefas, atualizar tarefas, fazer tarefas, mudar o prazo final das tarefas, mudar a dificuldade das tarefas, limpar a lista de tarefas e ordenar a lista de tarefas.

Diagrama de Classes do Trabalho:



Um Diagrama de Sequência do Trabalho:



GRUPO DO TRABALHO:

Guilherme Santos, Igor Braziel e Pablo Oliveira

Obrigado pela atenção!

Dúvidas?
Sugestões?