Trabalho Final

Relatório da Implementação da ACMERescue

Estudantes: Dalton Belman Albeche, Inácio Frick Pimentel, Stéfano De Paris Carraro

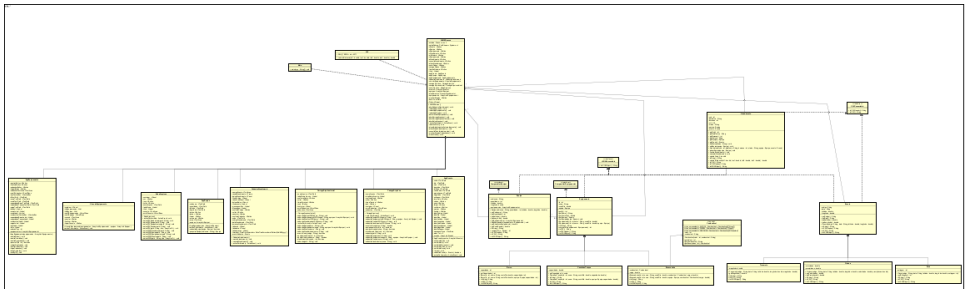
Disciplina: Programação Orientada a Objetos

Prof: Marcelo H. Yamaguti

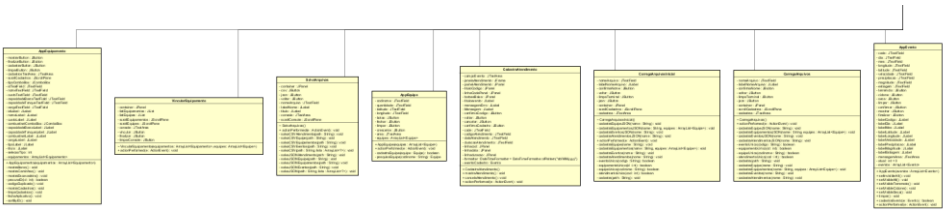
29 de novembro de 2023

1. Diagrama de classes:

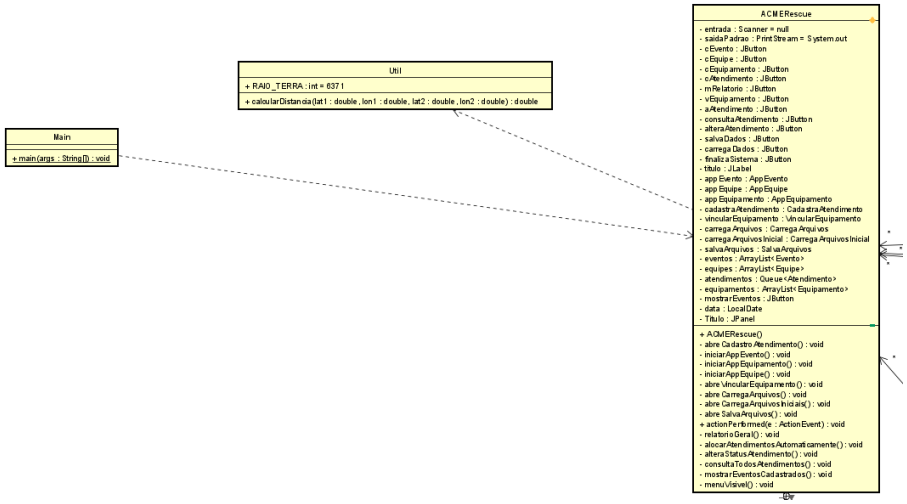
1.1 Foto Geral do Diagrama.



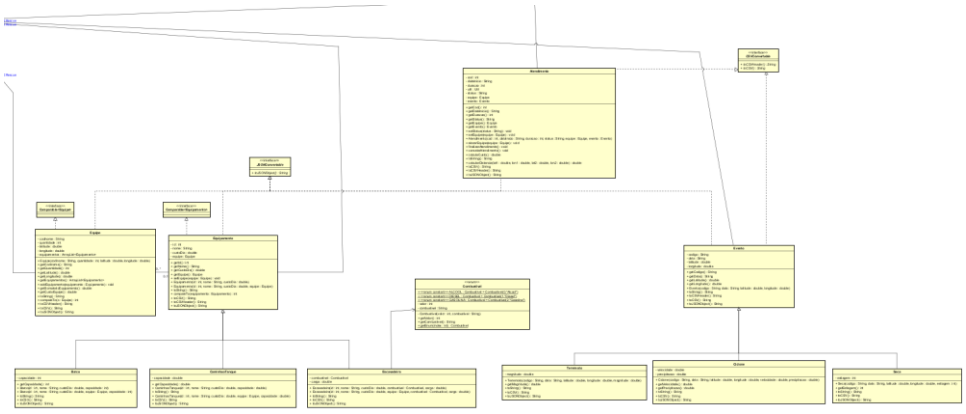
1.2 Foto das Inner Classes de AcmeRescue, usadas para facilitar passagem de parâmetros.



1.3 Foto da ACMERescue junto com Main e classe Util(Criada para calcular a distância).



1.4 Foto das Classes Equipe, Equipamento, Evento e Antendimento, junto com suas especializações e Interface implementadas como JSONConvertable, CSVConvertable e Comparable. Também é notável que as setas que saem para a equerda cima da foto são as mesmas que chegam no ACMERescue na foto anterior pela direita, com cardinalidade de *, isto quer dizer que ACMERescue guarda 1 ArrayList de cada uma delas.



Obs: Devido ao tamanho do diagrama creio que foi complicado demonstrar tudo por foto, portanto anexei o arquivo final .astah do diagrama dentro da pasta do trabalho, para caso tenha ficado alguma dúvida por foto.

2. **Coleções de dados**

O código faz uso de diversas coleções de dados feitas para armazenar informações sobre eventos, equipamentos, equipes e atendimentos. Cada tipo de coleção foi escolhido com base nas suas características e funcionalidades específicas, proporcionando eficiência e facilidade de implementação.

2.1 Listas

Usada para armazenar eventos, equipamentos e equipes. São úteis para o armazenamento de coleções de objetos. Foram instanciadas como ArrayList e estão implementadas no construtor da classe App “ACMERescue”.

Exemplo de instância de lista no construtor:

```
// Criação de listas para eventos, equipamentos e equipes
eventos = new ArrayList<>();
equipamentos = new ArrayList<>();
equipes = new ArrayList<>();
atendimentos = new LinkedList<>();
```

2.2 Filas

Usada para implementar a lista de atendimentos pendentes, esta abordagem permite uma atualização dinâmica à medida que os atendimentos são alocados. Ela é instanciada no método privado que aloca os atendimentos automaticamente “alocarAtendimentosAutomaticamente()”.

Exemplo de instância de lista no construtor:

```
// Instanciação da fila para atendimentos pendentes
Queue<Atendimento> atendimentosPendentes = new LinkedList<>();
```

2.3 JList e DefaultListModel

Componentes gráficos, como “JList”, são utilizados na interface gráfica do usuário para exibir dinamicamente a lista de equipes e equipamentos cadastrados. A utilização desses componentes proporciona uma experiência interativa para o usuário.

```
// Instanciação de DefaultListModel para preenchimento de JList
DefaultListModel<String> stringEquipamentos = new DefaultListModel<>();
DefaultListModel<String> stringEquipes = new DefaultListModel<>();
```

```
// Adição de elementos ao DefaultListModel utilizando for-each
stringEquipamentos.addElement(equipamento.getNome());
stringEquipes.addElement(equipe.getCodinome());

// Associação do DefaultListModel à JList
listEquipes = new JList<String>(stringEquipes);
listEquipamentos = new JList<String>(stringEquipamentos);
```

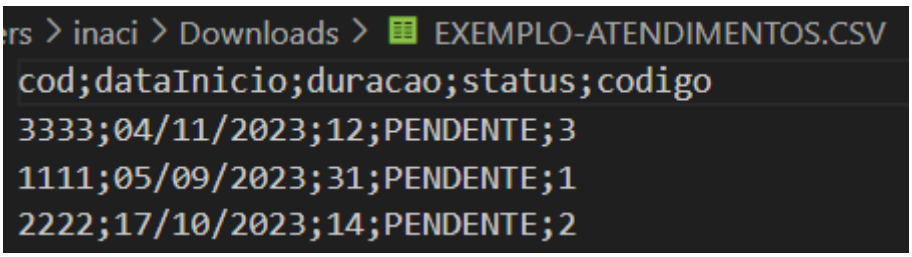
3. **Armazenamento de dados**

Optamos por utilizar os formatos de arquivo CSV e JSON para garantir eficiência e simplicidade.

3.1 Formato CSV

O formato CSV, foi a primeira opção implementada para leitura e salvamento de arquivos, devido à sua simplicidade e fácil adaptação para passar de CSV para objeto Java e vice-versa, já que consiste em apenas os atributos de forma textual separadas por um caractere “;”. Sendo assim necessitando apenas do split da linha em “;” e transformando cada um de seus atributos numa posição de um vetor de strings, tratando cada uma das String a partir dos metodos Parse e outros, para poder no fim instanciar um objeto.

// Exemplo do arquivo EXEMPLO-ATENDIMENTO.CSV:



3.2 Formato JSON

O formato JSON por sua vez, foi o escolhido para implementação relativa ao ponto adicional. No entanto, é notável que a manipulação do arquivo JSON é muito mais complexa do que a dos CSV, sendo que após um tempo de pesquisa a maioria dos materiais encontrados sobre leitura de arquivos JSON indicava a necessidade do uso de bibliotecas externas para processamento de maior qualidade e robustez.

Ademais, nosso grupo tomou o rumo mais difícil e decidiu tratar manualmente os arquivos JSON. Ao mesmo tempo, esse tratamento manual permitiu reusar uma boa parte do código do cadastro CSV, já que após o tratamento do arquivo, ficamos com um vetor de Strings em ambos e assim seguimos para os Parses e instanciar objetos em ambos.

Mesmo com todo reuso de código, o tratamento dos arquivos JSON, principalmente na parte da leitura, não deixou de ser um desafio, já que foram necessários diversos Replaces para transformar as linhas de JSON em atributos e finalmente objetos.

// Exemplo do arquivo EXEMPLO-ATENDIMENTO.JSON:

```
[
  {"cod": "3333", "dataInicio": "04/11/2023", "duracao": 12, "status": "PENDENTE", "codigo": "3"},
  {"cod": "1111", "dataInicio": "05/09/2023", "duracao": 31, "status": "PENDENTE", "codigo": "1"},
  {"cod": "2222", "dataInicio": "17/10/2023", "duracao": 14, "status": "PENDENTE", "codigo": "2"}
]
```

4. Observações

Leitor de arquivos mudado para guardar a situação do atendimento, caso de finalizado, pendente ou cancelado guarda o estado, apenas quando o status é executando ele troca para pendente, já que os arquivos não carregavam equipe e sem equipe o evento não pode estar como executando.

Se estiver dando algum problema quanto a gravação/leitura de arquivos, certifique-se de que a pasta aberta no IntelliJ é a mais externa “TrabalhoPOO”.