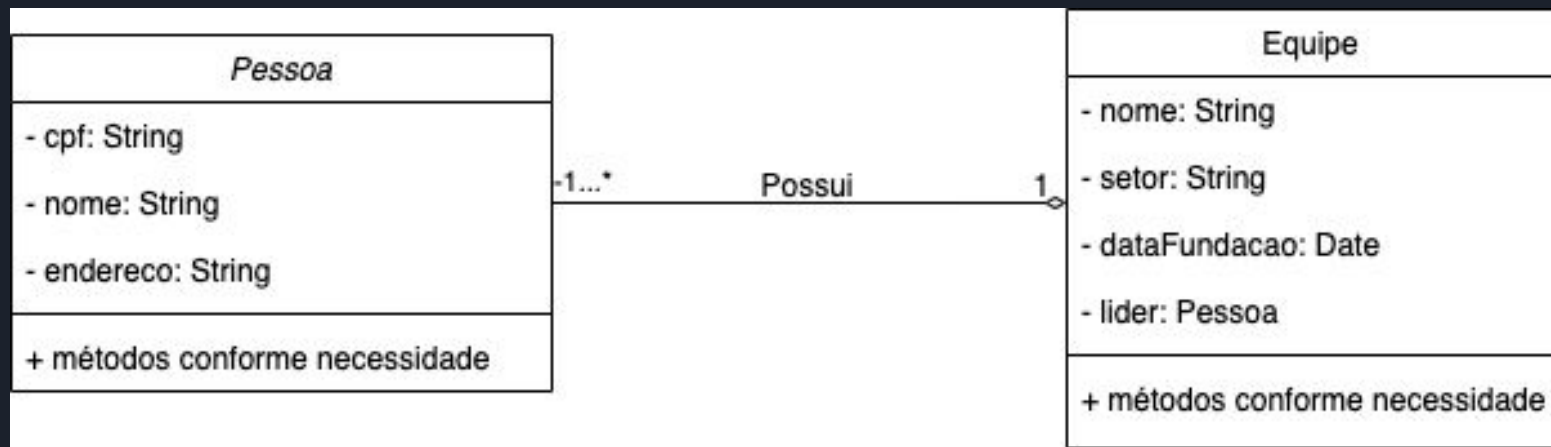




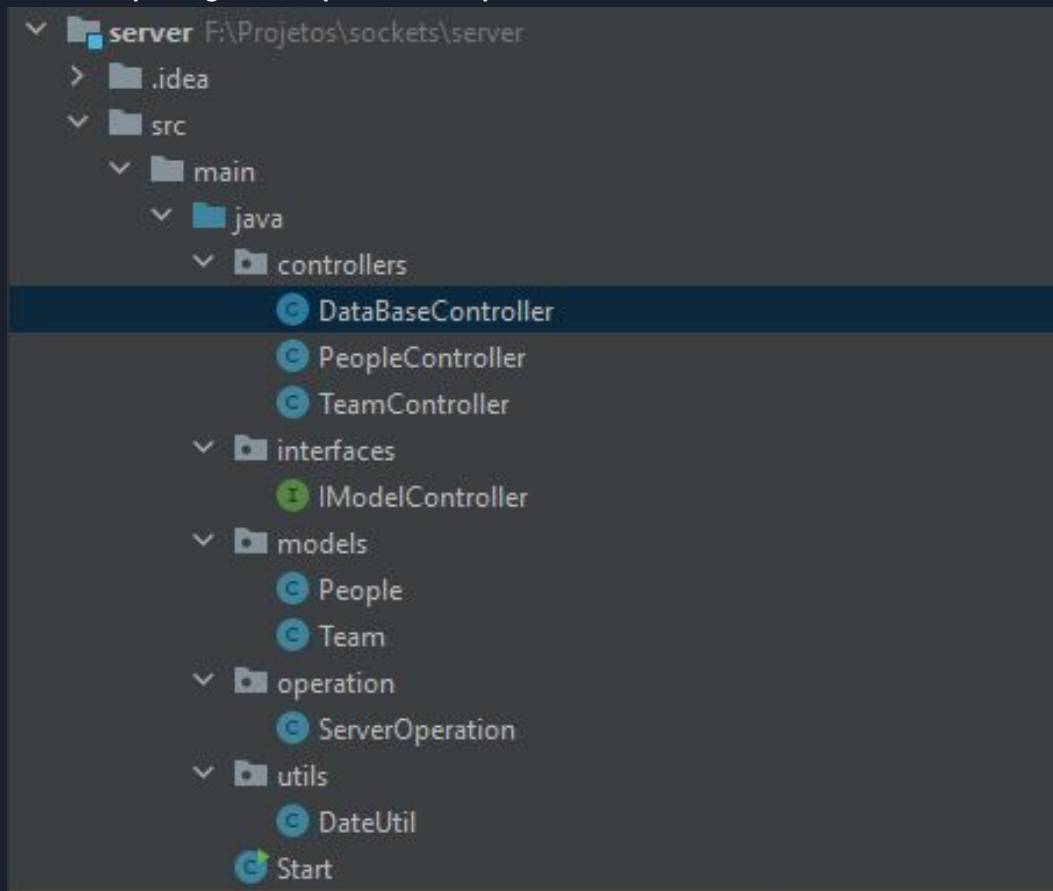
Sockets

Estudantes: André Cristen e Lucas Levi Gonçalves

Models




Estrutura do projeto (Server)





Código(Server)

```
public class Start {  
  
    public static void main(String[] args) throws Exception {  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Qual a porta que você deseja iniciar o servidor?");  
        new ServerOperation(scanner.nextInt());  
    }  
}
```



Código(Server)

```
public class ServerOperation {  
  
    1 usage  
    final String OPERATION_INSERT = "INSERT";  
    1 usage  
    final String OPERATION_UPDATE = "UPDATE";  
    1 usage  
    final String OPERATION_GET = "GET";  
    1 usage  
    final String OPERATION_DELETE = "DELETE";  
    1 usage  
    final String OPERATION_LIST = "LIST";  
    1 usage  
    final String OPERATION_ADD = "ADD";  
    1 usage  
    final String OPERATION_REMOVE = "REMOVE";  
  
    10 usages  
    PrintStream printStream;  
    2 usages  
    DataInputStream dataInputStream;
```



Código(Server)

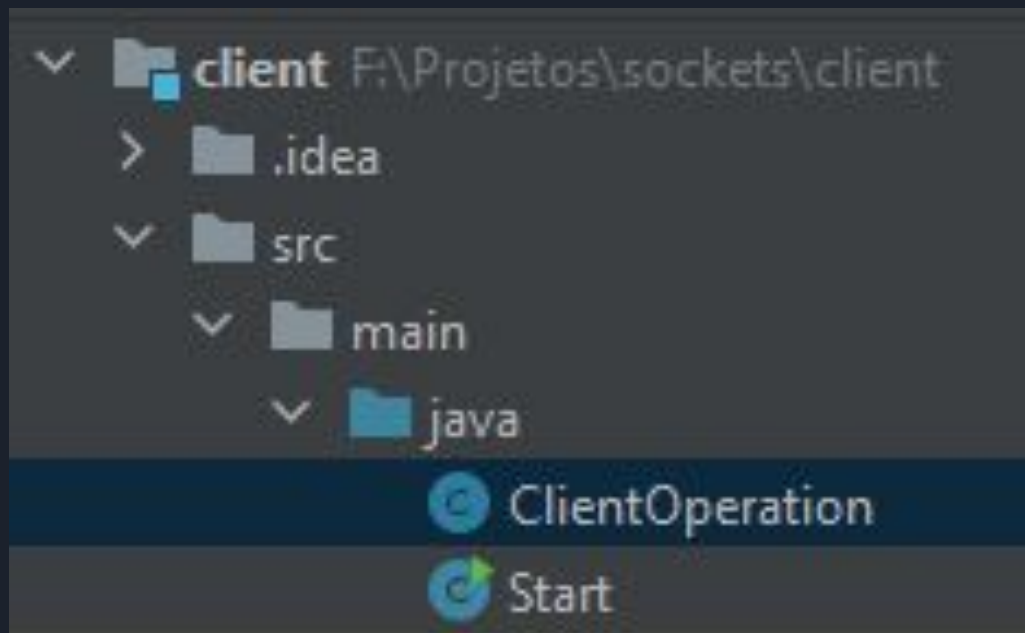
```
public ServerOperation(int port) throws Exception {  
    ServerSocket serverSocket = new ServerSocket(port);  
    System.out.println("Esperando conexões");  
    serverSocket.setReuseAddress(true);  
    Socket socket = serverSocket.accept();  
  
    System.out.println("Conexão estabelecida");  
    this.dataInputStream = new DataInputStream(socket.getInputStream());  
  
    boolean inOperation = true;  
    while (inOperation) {  
        String message = this.dataInputStream.readUTF();  
        String response = this.executeOperation(message);  
        System.out.println("Resposta: \n" + response);  
        socket.getOutputStream().write(response.getBytes());  
    }  
}
```

Código(Server)

```
private String executeOperation(String message) throws Exception {
    System.out.println("Requisição recebida: " + message);
    String response;
    HashMap<String, String> params = this.getParams(message);
    IModelController controller;
    switch (params.get("modelo")) {
        case "pessoa":
            controller = new PeopleController();
            break;
        case "equipe":
            controller = new TeamController();
            break;
        default:
            throw new Exception("Modelo não reconhecido.");
    }
}
```

```
switch (params.get("operacao")) {
    case OPERATION_INSERT:
        response = controller.insert(params);
        break;
    case OPERATION_UPDATE:
        response = controller.update(params);
        break;
    case OPERATION_GET:
        response = controller.get(params);
        break;
    case OPERATION_DELETE:
        response = controller.delete(params);
        break;
    case OPERATION_LIST:
        response = controller.list(params);
        break;
    case OPERATION_ADD:
        response = controller.add(params);
        break;
    case OPERATION_REMOVE:
        response = controller.remove(params);
        break;
    default:
        throw new Exception("Operação não reconhecida.");
}
return response;
```

Estrutura do projeto (Client)




Código(Client)

```
public class Start {  
    public static void main(String[] args) throws Exception {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Insira o ip do host:");  
        String host = s.next();  
        System.out.println("Insira a porta do host:");  
        int port = s.nextInt();  
        new ClientOperation(host, port);  
    }  
}
```



Código(Client)

```
public ClientOperation(String host, int port) throws Exception {  
    this.socket = new Socket(host, port);  
    this.stream = new DataOutputStream(this.socket.getOutputStream());  
    this.executeOperation();  
}
```



Código(Client)

```
private void executeOperation() throws Exception {  
    InputStream in = this.getSocket().getInputStream();  
    byte[] dadosBrutos = new byte[1024];  
    int qtdBytesLidos = 0;  
    while (qtdBytesLidos >= 0) {  
        qtdBytesLidos = in.read(dadosBrutos);  
        String response = new String(dadosBrutos, offset: 0, qtdBytesLidos);  
        System.out.println(response);  
        selectModelOperation();  
    }  
}
```

Código(Client)

```
private void selectModelOperation() throws Exception {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Selecione o modelo que deseja gerir:\n 1 - Pessoa\n 2 - Equipe\n 3 - Sair");
    int modelToHandle = scanner.nextInt();
    switch (modelToHandle) {
        case 1:
            selectDefaultOptionsOperation();
            break;
        case 2:
            selectDefaultOptionsOperation();
            selectEquipeOptionsOperation();
            break;
        case 3:
            throw new Exception("Fim da execução");
        default:
            System.out.println("Opção inválida pressione enter.");
            break;
    }
    int operation = scanner.nextInt();
    switch (modelToHandle) {
        case 1:
            handlePessoa(operation);
            break;
        case 2:
            handleEquipe(operation);
            break;
    }
}
```



Código(Client)

```
private void selectDefaultOptionsOperation() {  
    System.out.println("Por favor, escolha a operação: \n 1 - INSERT \n 2 - UPDATE\n 3 - DELETE \n 4 - GET \n 5 - LIST");  
}  
  
1 usage  
private void selectEquipeOptionsOperation() {  
    System.out.println(" 6 - ADD PESSOA\n 7 - REMOVE PESSOA \n");  
}
```

Código(Client)

```
private void handlePessoa(int option) throws IOException {
    Scanner scanner = new Scanner(System.in);
    String params = "";
    String operation = "";
    switch (option) {
        case 1 : {
            System.out.println("cpf");
            String cpf = scanner.next();
            System.out.println("nome");
            String nome = scanner.next();
            System.out.println("endereco");
            String endereco = scanner.next();
            params = "cpf=" + cpf + ";nome=" + nome + ";endereco=" + endereco;
            operation = OPERATION_INSERT;
            break;
        }
        case 2: {
            System.out.println("cpf");
            String cpf = scanner.next();
            System.out.println("nome");
            String nome = scanner.next();
            System.out.println("endereco");
            String endereco = scanner.next();
            params = "cpf=" + cpf + ";nome=" + nome + ";endereco=" + endereco;
            operation = OPERATION_UPDATE;
            break;
        }
    }
}
```



Código(Client)

```
String message = "modelo=pesoa;operacao=" + operation + ";" + params;  
this.getStream().writeUTF(message);
```

```
}
```

Código(Client)

```
private void handleEquipe(int option) throws IOException {
    Scanner scanner = new Scanner(System.in);
    String params = "";
    String operation = "";
    switch (option) {
        case 1: {
            System.out.println("Nome da equipe:");
            String nomeEquipe = scanner.next();
            System.out.println("cpf do lider:");
            String cpfLider = scanner.next();
            System.out.println("Setor:");
            String setor = scanner.next();
            System.out.println("Data de fundação (dd/mm/yyyy):");
            String dataFundacao = scanner.next();
            params = "nome=" + nomeEquipe + ";lider=" + cpfLider + ";setor="+setor+";dataFundacao="+dataFundacao;
            operation = OPERATION_INSERT;
            break;
        }
    }
}
```




Código(Client)

```
String message = "modelo=equipe;operacao=" + operation + ";" + params;  
this.getStream().writeUTF(message);
```



Sockets

Estudantes: André Cristen e Lucas Levi Gonçalves