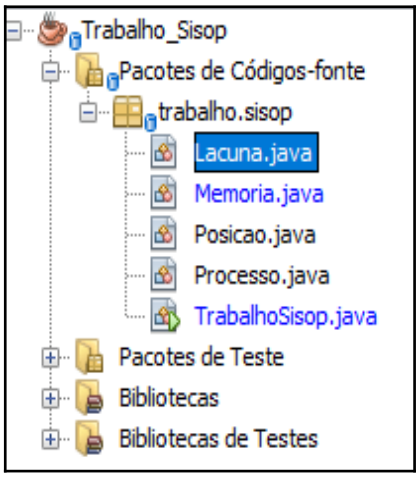
Relatório

Nome: Anderson Rodrigues (96614) e Jean Kistenmacher (80993)
Disciplina: Sistemas Operacionais

- **Estrutura do Projeto**



- **TrabalhoSisop.java:** Responsável pela inicialização da aplicação e por exibir o menu com as opções de algoritmo.
- **Memória.java:** Responsável por toda a lógica de simulação do gerenciamento de memória.
- **Posição.java:** Responsável por armazenar dados da posição do arquivo (pid e valor);
- **Processo.java:** Responsável por armazenar dados do processo (pid, quantidade de memória, inicio e fim da memória e se possui alguma instrução pendente)
- **Lacuna.java:** Responsável por armazenar dados de lacunas na memória.

- **TrabalhoSisop.java**
  - Exibe as opções de algoritmo de gerenciamento de partições no console.
  - Valida e captura a seleção do usuário.
  - Instância a classe Memória e envia por parâmetro o algoritmo escolhido pelo usuário e também o que foi “printado” no console, para que seja armazenado no arquivo de log.

```
public class TrabalhoSisop {
    public static void main(String[] args) {
        try {
            String console = "Escolha o Algoritmo de Gerenciamento de Partições: " + System.lineSeparator()
                    + "1: FIRST-FIT" + System.lineSeparator()
                    + "2: BEST-FIT" + System.lineSeparator()
                    + "3: WORST-FIT" + System.lineSeparator()
                    + "4: CIRCULAR-FIT" + System.lineSeparator();
            System.out.println(console);
            Scanner in = new Scanner(System.in);
            int valorDoConsole = in.nextInt();
            if (valorDoConsole > 4 || valorDoConsole < 1) {
                System.out.println("Algoritmo inválido!");
            } else {
                console += "ALGORITMO ESCOLHIDO = 4" + System.lineSeparator();
                Memoria memoria = new Memoria(valorDoConsole, console);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Memória.java**
  1. **construtor**
     - Responsável por inicializar as listas, verificar a existência dos arquivos de log e de memória, e iniciar leitura dos arquivos com os processos.
  2. **leitorDeProcessos**
     - Responsável por fazer a leitura dos dois arquivos de processos, preencher variáveis de inicio e fim do arquivo de memória, verificar instruções do processo, gerar logs, listar lacunas e também gerenciar o algoritmo escolhido pelo usuário.
  3. **verificaGeraArquivoMemoria**
     - Verifica se existe o arquivo de memória criado, caso não existir, cria.

4. **preencheMetadeDoArquivo**
    - Responsável por ocupar metade do arquivo de memória

5. **geraDados**
    - Gera conteúdo para preencher metade do arquivo de memória.
6. **preencheInicioFimDaMemoria**
    - Responsável por buscar a posição de início e fim do arquivo de memória.
7. **carregaVariaveisParaLeituraAndEscrita**
    - Responsável por preencher as variáveis que serão utilizadas para a leitura e escrita do arquivo de memória
8. **encerraEscritaNoArquivo**
    - Responsável por encerrar a leitura do arquivo e escrever no arquivo de memória
9. **percorreAndPreecheArquivoByFgLacuna**
    - Responsável por preencher o arquivo de memória ou deixar lacunas.
10. **verificaProcesso**
    - Responsável por verificar as instruções do processo, verificar se é uma ES, SW, LW ou se o processo já finalizou e adicionar na fila.
11. **bestFit**
    - Responsável por encontrar o melhor espaço para o processo
12. **firstFit**
    - Responsável por colocar o processo no primeiro espaço que ele couber
13. **worstFit**
    - Responsável por colocar o processo pior espaço, ou seja, no maior espaço.
14. **circularFit**
    - Responsável por colocar o processo no espaço, seguindo a sequência do ultimo processo.
15. **atualizaProcessoAndLacunaLista**
    - Responsável por atualizar onde o processo inicia e acaba, e também por atualizar/remover a lacuna.
16. **listaLacunas**
    - Responsável por listar no console todas as lacunas
17. **verificaGeraArquivoLog**
    - Verifica se existe o arquivo de log criado, caso não existir, cria.
18. **GeraLog**
    - Responsável por escrever o valor que recebeu no arquivo de log

```
Memoria(Integer algoritmo, String logMenu) throws IOException {...13 linhas

private void leitorDeProcessos() throws FileNotFoundException, IOException {

private boolean verificaGeraArquivoMemoria() throws IOException {...8 linhas

private void preencheMetadeDoArquivo() throws IOException {...7 linhas }

private String geraDados() {...7 linhas }

private void preencheInicioFimDaMemoria() throws FileNotFoundException, IOEx

private void carregaVariaveisParaLeituraAndEscrita() throws IOException {...

private void encerraEscritaNoArquivo() throws IOException {...6 linhas }

private void percorreAndPreecheArquivoByFgLacuna(Processo processo, boolean

private void verificaProcesso(String[] data, Processo processo) throws IOExc

private Processo bestFit(Processo processo) {...16 linhas }

private Processo firstFit(Processo processo) {...9 linhas }

private Processo worstFit(Processo processo) {...16 linhas }

private Processo circularFit(Processo processo) {...14 linhas }

private Processo atualizaProcessoAndLacunaLista(Processo processo, Integer i

private void listaLacunas() throws IOException {...15 linhas }

private void verificaGeraArquivoLog() throws IOException {...7 linhas }

private void geraLog(String msg) throws IOException {...8 linhas }
```

- **Lacuna.java**
  - Responsável por armazenar a quantidade de espaço, onde inicia e onde acaba as lacunas.

```
public class Lacuna {

    private Integer qtd;
    private Integer posInicio;
    private Integer posfim;

    Lacuna(int qtd, Integer posInicio, Integer posFim) {...5 linhas }

    public Integer getQtd() {...3 linhas }

    public void setQtd(Integer qtd) {...3 linhas }

    public Integer getPosInicio() {...3 linhas }

    public void setPosInicio(Integer posInicio) {...3 linhas }

    public Integer getPosfim() {...3 linhas }

    public void setPosfim(Integer posfim) {
        this.posfim = posfim;
    }

}
```

- **Posicao.java**
  - Responsável por armazenar o valor e qual processo está armazenado naquela posição do arquivo.

```
public class Posicao {

    private String pid;
    private String value;

    Posicao(String pid) {...4 linhas }

    public String getPid() {...3 linhas }

    public void setPid(String pid) {...3 linhas }

    public String getValue() {...3 linhas }

    public void setValue(String value) {...3 linhas }

}
```