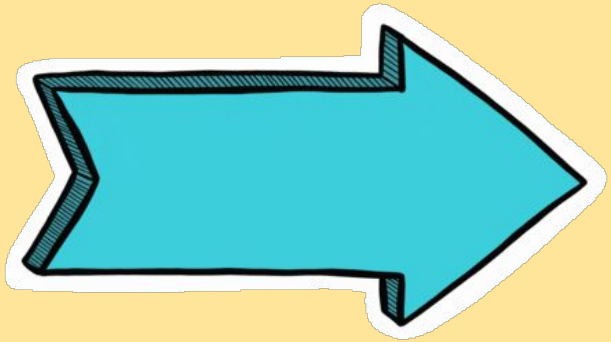


Controle de Férias

Grupo 5

Bruno Marques
Daniella Lira
Janaina Mai
Lucas Ivan

Backlog do produto

Controle de férias

Colaborador “comum”

Consulta de saldo

Consulta de histórico de férias realizadas

Solicitação de um período de férias específico

~~Envio de mensagem ao setor desejado (financeiro/RH) com assunto referente às férias~~

Pedido de venda de férias

Colaborador líder

Consulta de férias de um colaborador de sua equipe

RH

Verificação de solicitações (chamados de colaboradores)

Conceder férias (individual/coletiva) à um colaborador

Edição da concessão de férias de um colaborador

Consultar todos os colaboradores (por filtros)

Financeiro

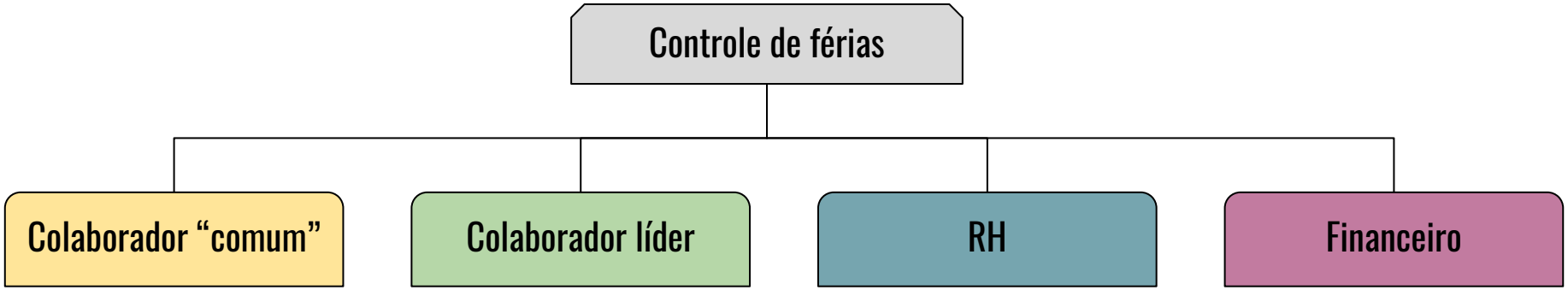
Consultar todos os colaboradores com opção de filtros

Legenda:

Feito

Não foi feito

~~Retirado~~



Consulta de saldo

Consulta de histórico de férias realizadas

Solicitação de um período de férias específico

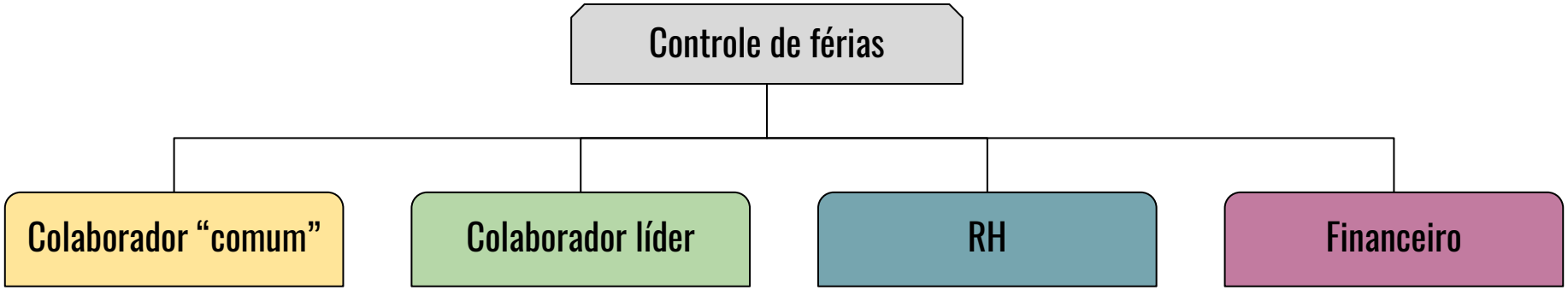
~~Envio de mensagem ao setor desejado (financeiro/RH) com assunto referente às férias~~

Pedido de venda de férias

O sistema recebe o colaborador e verifica se ele possui saldo de férias positivo.

Se sim, o sistema retorna quantidade de saldo de férias do colaborador.

Se não, informa que não existe saldo positivo de férias.



Consulta de saldo

Consulta de histórico de férias realizadas

Solicitação de um período de férias específico

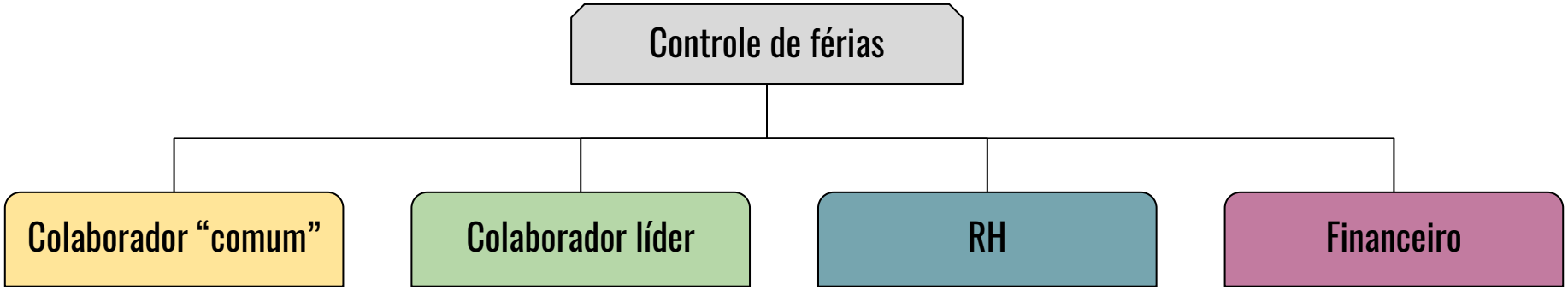
~~Envio de mensagem ao setor desejado (financeiro/RH) com assunto referente às férias~~

Pedido de venda de férias

O sistema recebe o colaborador e verifica se ele possui registro de férias realizadas.

Se sim, o sistema retorna relação de períodos de férias tiradas, contendo o tipo: férias vendidas (valor), férias de direito (quantidade de dias, e data de início e fim)

Se não, o sistema informa que não existe registro de férias realizadas.



Consulta de saldo

Consulta de histórico de férias realizadas

Solicitação de um período de férias específico

~~Envio de mensagem ao setor desejado (financeiro/RH) com assunto referente às férias~~

Pedido de venda de férias

O sistema recebe o colaborador e verifica se ele possui saldo de férias disponível.

Se não tiver, o sistema informa que o colaborador não possui saldo disponível.

Se tiver, o sistema retorna a quantidade de dias disponível.

O sistema recebe o tipo de férias total (recebe a data de início) ou parcial (recebe a data de início e data final).

O sistema irá verificar se a data de início informada é de pelo menos 10 dias após a data em que o chamado está sendo feito.

O que foi feito?

Solicitação de um período de férias específico

```java
/**
 * Verifica intervalo entre data da solicitação e data de início das férias
 *
 * Verifica se a data de início das férias solicitada é 10 dias após a data de solicitação, retornando verdadeiro caso seja.
 *
 * @param dataInicioFerias LocalDateTime - data de início de férias solicitada
 * @return boolean, de acordo com a condição (maior que 10 dias)
 */
public static boolean verificarDataSolicitacao(LocalDateTime dataInicioFerias) {
    boolean intervaloSuperior10dias = LocalDateTime.now().until(dataInicioFerias, ChronoUnit.DAYS) > 10? true : false;
    return intervaloSuperior10dias;
}
```

```java
@Test
public void verificaSeIntervaloEntreDataSolicitacaoEDataInicioFeriasEhMaiorQue10Dias() {
    boolean intervaloEntreSolicitacaoEInicio = Main.verificarDataSolicitacao(LocalDateTime.now().plusDays(20));
    assertTrue(intervaloEntreSolicitacaoEInicio);
}

@Test
public void verificaSeIntervaloEntreDataSolicitacaoEDataInicioFeriasEhMenorQue10Dias() {
    boolean intervaloEntreSolicitacaoEInicio = Main.verificarDataSolicitacao(LocalDateTime.now().plusDays(7));
    assertFalse(intervaloEntreSolicitacaoEInicio);
```

O que foi feito?

Solicitação de um período de férias específico

```
 * Verifica intervalo entre data inicio e data fim férias parciais
 *
 * Verifica se o intervalo entre a data de início e data de fim, considerando férias parciais, é inferior a 20 dias, retornando
 * verdadeiro caso seja.
 *
 * @param dataInicioFerias LocalDateTime - data de início de férias solicitada pelo colaborador
 * @param dataFimFerias LocalDateTime - data do fim das férias solicitada pelo colaborador
 * @return boolean, de acordo com a condição (inferior a 20 dias)
 */
public static boolean verificarIntervaloFeriasParciais(LocalDateTime dataInicioFerias, LocalDateTime dataFimFerias) {
    boolean intervaloInferior20dias = dataInicioFerias.until(dataFimFerias, ChronoUnit.DAYS) < 20? true : false;
    return intervaloInferior20dias;
}
```

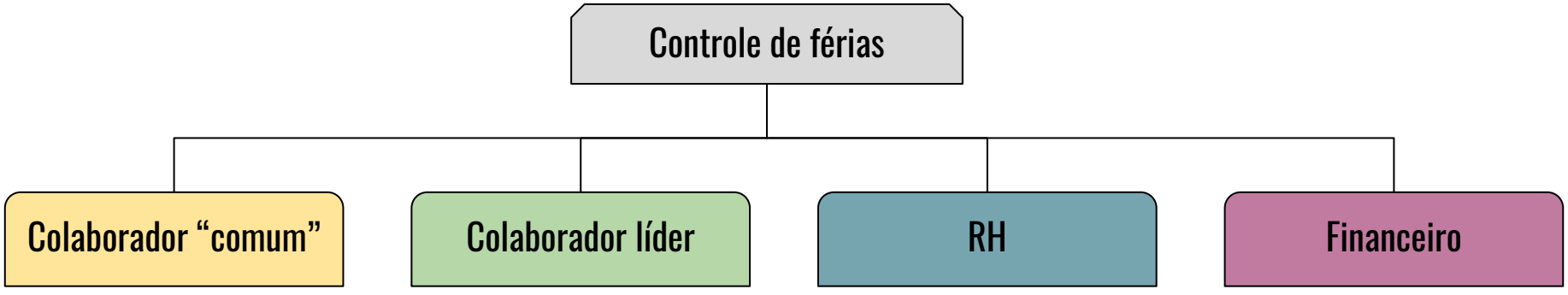
```
@Test
public void verificaSeIntervaloEntreDataInicioEDataFimFeriasEhMenorQue20Dias  () {
    boolean intervaloEntreSolicitacaoEInicio  = Main.verificarIntervaloFeriasParciais (LocalDateTime.of(2021, 3, 29, 0, 0),
LocalDateTime.of(2021, 4, 10, 0, 0));
    assertTrue (intervaloEntreSolicitacaoEInicio );
}

@Test
public void verificaSeIntervaloEntreDataInicioEDataFimFeriasEhMaiorQue20Dias  () {
    boolean intervaloEntreSolicitacaoEInicio  = Main.verificarIntervaloFeriasParciais (LocalDateTime.of(2021, 3, 29, 0, 0),
LocalDateTime.of(2021, 5, 1, 0, 0));
    assertFalse (intervaloEntreSolicitacaoEInicio );
}
```

O que foi feito?

Solicitação de um período de férias específico

```java
/**
 * Calcula data fim das férias totais
 *
 * Calcula a data fim das férias totais com base na data início das férias solicitada pelo colaborador.
 * O cálculo é realizado considerando que a data fim será 30 dias após a data início.
 *
 * @param dataInicioFerias LocalDateTime - data de início de férias solicitada pelo colaborador
 * @return data fim das férias totais
 */
public static LocalDateTime calculaDataFimFeriasTotais(LocalDateTime dataInicioFerias) {
    LocalDateTime dataFimFerias = dataInicioFerias.plusDays(30);
    return dataFimFerias;
}
```

```java
@Test
public void verificaSeIntervaloEntreDataInicioEDataFimParaFeriasTotaisEhIgualA30Dias() {
    LocalDateTime dataInicioFeriasTotais = LocalDateTime.of(2021, 3, 29, 0, 0);
    LocalDateTime dataFimFeriasTotais = Main.calculaDataFimFeriasTotais(dataInicioFeriasTotais);
    boolean intervaloEntreDataInicioEFim = dataInicioFeriasTotais.until(dataFimFeriasTotais, ChronoUnit.DAYS) == 30? true : false;
    assertTrue(intervaloEntreDataInicioEFim);
}
```



Consulta de férias de um colaborador de sua equipe

O sistema recebe o usuário e verifica se este possui colaboradores em sua equipe

Caso haja, o sistema retorna uma lista do nome dos colaboradores contendo: tipo de férias, data de início e fim, solicitação de férias em andamento

Caso não haja, o sistema informa que não há cadastros de colaboradores na equipe do usuário

O que foi feito?

Consulta de férias de um colaborador de sua equipe

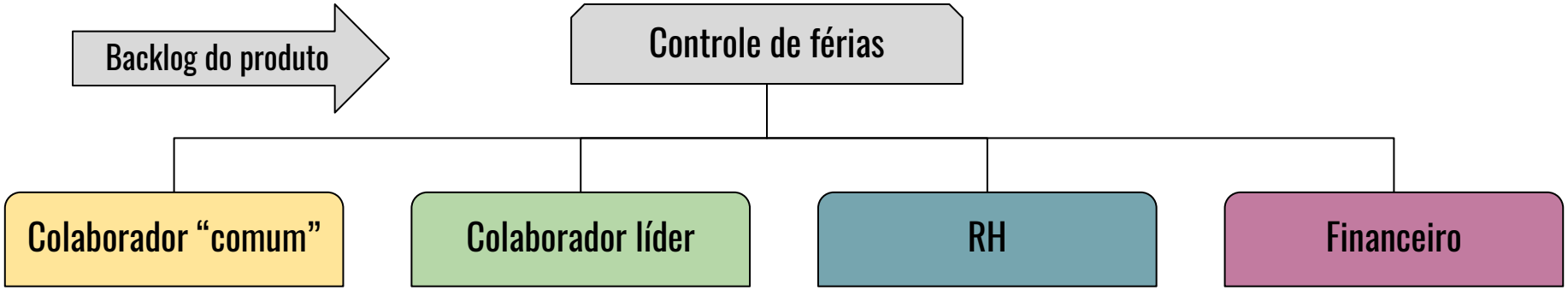
```java
/**
 * Consulta férias de membros da equipe
 * Após verificar se o usuário possui colaboradores em sua equipe,
 * o sistema retorna quais colaboradores são da equipe e o saldo de férias de cada um
 *
 * @author Bruno Marques
 * @param listaDeColaboradores int[] recebe vetor com lista de colaboradores
 * @param saldoDeFerias int[] recebe vetor com saldo de férias por colaborador
 * @return uma lista de colaboradores da equipe com saldo de férias de cada um
 */
    public static ArrayList<Integer>  consultaSituacaoDeFeriasDaEquipe(int[] listaDeColaboradores,
            int[] saldoDeFerias) {
        ArrayList<Integer> lista = new ArrayList<Integer>();
        if (consultarColaboraoresDaEquipe(recebeColaboradores)) {
            System.out.println("Você possui " + listaDeColaboradores.length + " colaboradores em sua equipe.");
            for (int c = 0; c < listaDeColaboradores.length; c++) {
                int colab = listaDeColaboradores[c];
                lista.add(saldoDeFerias[colab]);
                System.out.println("O colaborador de id " + listaDeColaboradores[c] + " possui " +
                saldoDeFerias[c] + " dias de férias." );
            }
        } else {
            System.out.println("O usuário não possui colaboradores em sua equipe");
        }
        return lista;
    }
```

Console:
Você possui 3 colaboradores em sua equipe.
O colaborador de id 0 possui 30 dias de férias.
O colaborador de id 1 possui 25 dias de férias.
O colaborador de id 4 possui 40 dias de férias.

O que foi feito?

Consulta de férias de um colaborador de sua equipe

```
@Test
public void testeConsultaSituacaoDeFeriasDaEquipe() {
    ColaboradorLider ColaboradorLider = new ColaboradorLider();
    ArrayList<Integer> saldos = ColaboradorLider.consultaSituacaoDeFeriasDaEquipe(listaDeColaboradores, saldoDeFerias);
    for(int i = 0; i < saldos.size(); i++) {
        int saldoRecebido = saldos.get(i);
        int j = listaDeColaboradores[i];
        int saldoEsperado = saldoDeFerias[j];
        assertEquals(saldoRecebido, saldoEsperado);
    }
```



O sistema retorna uma lista com todos os chamados que possuem status de pendente.

Caso o usuário queira buscar por status (em atraso, finalizado, deferido) o sistema retorna a lista de chamados conforme status desejado.

O que foi feito?

Verificação de solicitações - chamados de colaboradores

Criamos os recursos para retornar os métodos

```
public String[][] chamados = new String[4][3];
```

```
public static String informaTipoChamado(int tipoChamado) {
        String nomeChamado = "Chamado";
        switch (tipoChamado) {
        case 0:
            nomeChamado = "Chamado pendente";
            break;
        case 1:
            nomeChamado = "Chamado deferido";
            break;
        case 2:
            nomeChamado = "Chamado finalizado";
            break;
        case 3:
            nomeChamado = "Chamado em atraso";
            break;
        }
        return nomeChamado;
```

```
public static String informaNãoExistemChamados() {
        return "Não existem chamados com o status solicitado.";
    }
```

O que foi feito?

Verificação de solicitações - chamados de colaboradores

Criamos os métodos

```
public static boolean
verificaSeExistemChamados(String[] listaChamados) {
        int quantidadeChamados = 0;
        boolean existemChamados = false;
        for (String chamado : listaChamados) {
            quantidadeChamados++;
        }

        if (quantidadeChamados > 0) {
            existemChamados = true;
        }

        return existemChamados;
```

Percorremos os dados do vetor com retorno da mensagem

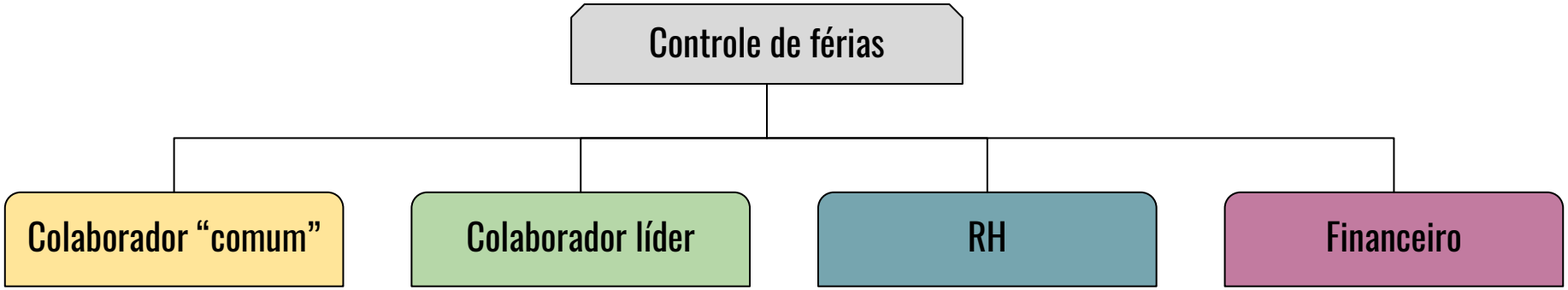
```
        chamados[2][0] = "chamado 1";
        chamados[2][1] = "chamado 2";
        chamados[2][2] = "chamado 3";

        chamados[3][0] = "chamado 1";
        chamados[3][1] = "chamado 2";
        chamados[3][2] = "chamado 3";
    }
    public  void consultaC
hamadosPendentes() {
        // usuário clicou em consultar chamados
pendentes
        int tipoChamado = 0;
        String[] chamados = getChamados(tipoChamado);
        if (verificaSeExistemChamados(chamados)) {

System.out.println(informaTipoChamado(tipoChamado));
            for (String chamado : chamados) {
                System.out.println(chamado);
            }
        } else
            informaNãoExistemChamados();
    }
}
```

O que foi feito?

Verificação de solicitações - chamados de colaboradores

Implementamos com assertEquals para testar a igualdade entre os chamados (esperado X retornado.)

```
@Test
    public void testeGetChamados() {
        StatusDeChamados main = new StatusDeChamados();
        main.criarChamados();

        int tipoChamado = 0;
        String[] retornoChamados = main.getChamados(tipoChamado);
        assertEquals("chamado 01", retornoChamados[0]);
        assertEquals("chamado 02", retornoChamados[1]);
        assertEquals("chamado 03", retornoChamados[2]);

    }
```



O sistema recebe um colaborador (objeto) ou mais (lista de objetos) que terão suas férias editadas.

O sistema recebe o período cadastrado que será editado.

O sistema permite a inserção de nova data de início e/ou fim.

~~O sistema verifica se a quantidade de dias permanece a mesma a anterior.~~

Alterado

O sistema valida a quantidade de dias do novo período: precisa ser menor ou igual ao período a ser editado, e diferente de zero.

Verificação de solicitações (chamados de colaboradores)

Conceder férias (individual/coletiva) à um colaborador

Edição da concessão de férias de um colaborador

Consultar todos os colaboradores

O que foi feito?

Edição da concessão de férias de um colaborador

```java
/**
     * Retorna quantidade de dias
     *
     * Retorna a quantidade de dias em formato long, a partir das datas de início e término informadas.
     *
     * @param inicio  Data de início.
     * @param termino Data de término.
     * @return
     */
    public static long retornarIntervaloEmDiasEntreAsDatas(LocalDate inicio, LocalDate termino) {
        long dias = ChronoUnit.DAYS.between(inicio, termino);
        return dias;
    }
```

```java
@Test
    public void testeRetornarIntervaloEmDiasEntreAsDatas() {
        LocalDate dataInicio = LocalDate.of(2020, 01, 01);
        LocalDate dataTermino = LocalDate.of(2020, 01, 11);

        long dias = Main.retornarIntervaloEmDiasEntreAsDatas(dataInicio, dataTermino);
        assertEquals(dias, 10);     }
```

O que foi feito?

Edição da concessão de férias de um colaborador

```java
/**
     * Valida o novo periodo de concessão de férias.
     *
     * Verifica se o período é valido com base na quantidade de dias do período atual e do período novo.
     *
     * @param periodoAtual Quantidade de dias do período atual a ser alterado.
     * @param dataInicio Data de início do período.
     * @param dataTermino Data de término do período.
     * @return
     */
    public static boolean novoPeriodoEhValido(int periodoAtual, LocalDate dataInicio, LocalDate dataTermino) {
        boolean periodoValido = true;

        long periodoNovo = ChronoUnit.DAYS.between(dataInicio, dataTermino);
        if (periodoAtual < periodoNovo || periodoNovo == 0) {
            periodoValido = false;
        }
        return periodoValido;
    }
```

O que foi feito?

Edição da concessão de férias de um colaborador

```
@Test
    public void testeNovoPeriodoEhValido() {
        int periodoAtual = 5;
        LocalDate dataInicio = LocalDate.of(2021, 01, 01);
        LocalDate dataTermino =  LocalDate.of(2021, 01, 11);

        boolean valido = Main.novoPeriodoEhValido(periodoAtual, dataInicio, dataTermino);
        assertEquals(valido, false);
    }
```

O que foi feito?

Edição da concessão de férias de um colaborador

```
static int[] feriasTiradas = { 10, 15, 20, 30 };
```

```
/**
     * Altera período de concessão de férias parciais.
     *
     * Altera o período de concessão de férias através da nova data de início e
     * término informadas.
     *
     * @param idPeriodo       ID do período a ser alterado.
     * @param novaDataInicio  Nova data de início do período.
     * @param novaDataTermino Nova data de término do período.
     * @return
     */
    public static void alterarPeriodoDeFeriasParciais(int idPeriodo, LocalDate novaDataInicio,
            LocalDate novaDataTermino) {
        long novoPeriodo = retornarIntervaloEmDiasEntreAsDatas(novaDataInicio, novaDataTermino);
        feriasTiradas[idPeriodo] = (int) novoPeriodo;
    }
```

O que foi feito?

Edição da concessão de férias de um colaborador

```
@Test
    public void testeAlterarPeriodoDeFeriasParciais() {
        int idPeriodo = 0;

        LocalDate novaDataInicio = LocalDate.of(2021, 01, 01);
        LocalDate novaDataTermino = LocalDate.of(2021, 01, 11);

        Main.feriasTiradas[idPeriodo] = 5;

        Main.alterarPeriodoDeFeriasParciais(idPeriodo, novaDataInicio, novaDataTermino);

        assertEquals(Main.feriasTiradas[idPeriodo], 10);

    }
```

Retrospectiva da Sprint

O que deu certo

O que deu errado

O que aprendemos

- Construção do backlog do produto

- Conclusão do backlog de sprint
  - Por quê? R: Distribuição de tarefas adequadas de acordo com número de integrantes e suas capacidades
  - O que fez diferença? R: Trabalho e aprendizado em dupla/equipe

- Implementação dos métodos com os recursos limitados (ex: sem orientação a objetos)
  - Por quê? R: Abstração de dados

Retrospectiva da Sprint

O que deu certo | **O que deu errado** | O que aprendemos

- Achar que RH não paga férias ao colaborador e que precisava ser feito um pedido ao setor financeiro: inicialmente começamos a pesquisar sobre RH e férias, mas o solicitado foi para focar nas histórias, isso posteriormente resultou em uma informação errada e a criação de várias histórias que foram canceladas após percebermos que a informação não era da nossa atribuição.

- Redundância de métodos: fizemos métodos que traziam o mesmo resultado ou ação.

- Alguns checklists ficaram mal detalhados: posteriormente o que deveria ser 1 método acabou se tornando 3.

Retrospectiva da Sprint

O que deu certo | O que deu errado | **O que aprendemos**

- Trabalhar em equipe: pois separamos as tarefas de acordo com as habilidades de cada um, a fim de compartilhar conhecimento e elevar o aprendizado, isso só deu certo devido a colaboração de todos e à programação em conjunto.

- Programar em conjunto: o resultado é melhor e mais rápido, conseguimos nos atentar a detalhes que sozinhos passariam despercebidos.

- Precisamos cuidar melhor do projeto como um todo: pois a partir do momento em que cada um começou a desenvolver seus métodos, ocorreram redundâncias e foi preciso posteriormente corrigi-las.

O que faremos na próxima?

Fazer mais:

- ❏ Organizar mais backlog da sprint (evitar sobreposição redundante).
- ❏ Focar na tarefa em si para não fazer alterações que resultarão em mais trabalho.
- ❏ Contextualizar mais sobre o tema

Fazer menos:

- ❏ Refatoração de métodos na hora errada
- ❏ Tentar escrever um código “perfeito”, que não sofrerá alterações

Parar de fazer:

- ❏ Ter insegurança
- ❏ Síndrome do Impostor

Continuar fazendo:

- ❏ Programação em conjunto
- ❏ Contribuição entre o time
- ❏ Disposição para ajudar

Começar a fazer:

- ❏ Acompanhamento das tarefas como um todo
- ❏ Ordenar tarefas por prioridade
- ❏ Reconhecimento do trabalho
- ❏ Entender como funciona o GIT

Obrigado pela atenção!