Considere a existência dos seguintes tipos:

```
public interface Job{// Tarefa
   String getName()          // Nome da tarefa
   LocalDate getStartDate(); // Data de início
   LocalDate getFinishDate();// Data de fim
}
```

```
public class LocalDate
    implements Comparable<LocalDate>{
 public static LocalDate now(){…}
 // text string such as 2023-01-13
 public static
       LocalDate parse(String txt){…}
 public String toString(){…}
   …
}
```

```
public interface BiPredicate<E1, E2> {
 // Retorna true se e1 e e2 satisfazem o predicado
 boolean test(E1 e1, E2 e2);
}
```

```
public interface Consumer<E> {
  // Executa a ação sobre elem.
  void accept(E elem);
}
```

Grupo I

a) [3] Realize o método público estático com a seguinte assinatura:

```
void forEachIf( Iterable<Job> jobs, BiPredicate<LocalDate, LocalDate> pred,
                                    Consumer<Job> action)
```

Que execute a ação **action** sobre as tarefas da sequência **jobs** cujas datas de início e término satisfazem o predicado **pred**.

b) [3] Utilizando o método da alínea a) e implementando um **BiPredicate** e um **Consumer**, realize um método público estático com a seguinte assinatura:

```
SortedSet<Job> jobsInExecution( Collection<Job> jobs )
```

Que produz e retorna o conjunto de tarefas, contidas na coleção **jobs**, que estão em execução, isto é, a data corrente é superior à inicial e inferior à final. O conjunto produzido deve ficar ordenado de forma crescente de datas de início e para a mesma data por nomes das tarefas (o nome da tarefa é único). Use o método estático **now** da classe **LocalDate** para obter a data corrente.

c) [3] Realize o método público estático com a seguinte assinatura:

```
Collection<Job> startAfter(SortedMap<LocalDate,Set<Job>> jobsPerDate,
                           LocalDate minimum)
```

Que produza e retorne uma coleção com as tarefas, existentes no contentor associativo ordenado **jobsPerDate**, cuja data de início seja posterior a **minimum**. No contentor associativo as chaves são a data de início e o valor associado é o conjunto de tarefas que se iniciam nessa data. O contentor associativo está ordenado de forma decrescente pelas datas de início, isto é, as datas mais recentes estão no início.

Grupo II

```
public interface BiConsumer<E1, E2> {
  // Executa a ação sobre e1 e e2.
  void accept(E1 e1, E2 e2);
}
```

a) [4] Realize um método público estático com a seguinte assinatura:

```
int copyJobs(String pathnameIn,LocalDate dt,
             BiConsumer<String,LocalDate> action) throws IOException
```

Que leia o ficheiro de texto de nome **pathnameIn**, onde cada linha contém a identificação de uma tarefa e as datas de início e término *(<name>: <startDate> <finishDate>)*, e execute a ação **action** sobre as tarefas cuja data inicial é igual a **dt**. Ao método **accept**, evocado sobre **action**, deve ser passado o nome da tarefa e a data final.

O método **copyJobs** retorna o número de tarefas cuja data inicial é **dt**.

Use o método estático **parse** da classe **LocalDate** que recebendo por parâmetro uma *string* que representa uma data (ex. 2023-01-13), retorna a respetiva **LocalDate**.

jobs.txt
```
1º Trabalho: 2022-09-21 2022-10-30
1º Teste: 2022-11-08 2022-12-14
2º Trabalho: 2022-11-02 2022-11-02
3º Trabalho: 2022-12-15 2023-01-29
Exame de Época Normal: 2023-01-13 2023-01-13
2º Teste: 2023-01-13 2023-01-13
Exame de Época Recurso: 2023-01-30 2023-01-30
Repetição do 1º Teste: 2023-01-30 2023-01-30
Repetição do 2º Teste: 2023-01-30 2023-01-30
```

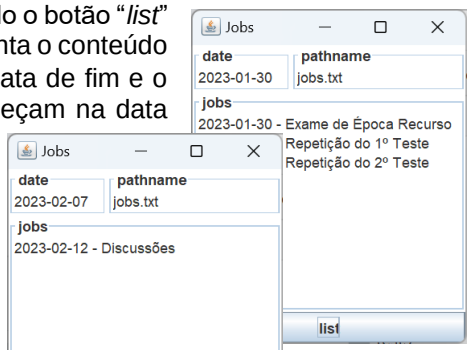
b) [2] Utilizando o método da alínea a) e implementado um **BiConsumer** realize um método público estático:

```
void copyTodayJobs(String pathnameIn, String pathnameOut) throws IOException
```

Que leia do ficheiro de texto de nome **pathnameIn**, onde cada linha contém a identificação de uma tarefa e as duas datas, e produza um ficheiro de texto de nome **pathnameOut** contendo o nome das tarefas que se iniciam na data corrente. Use o método estático **now** da classe **LocalDate** para obter a data corrente.

```
Exame de Época Normal
2º Teste
```

c) [5] Faça uma aplicação com o aspeto da figura. Quando for premido o botão "*list*" deve ser chamado o método **copyJobs** da alínea a), tendo em conta o conteúdo das caixas de texto. Na área de texto "*jobs*" deve ser escrito a data de fim e o respetivo nome das tarefas que constam no ficheiro e que começam na data presente na caixa de texto.
Em caso de erro no acesso ao ficheiro deve ser chamado o método estático **JOptionPane.showMessageDialog** passando como 1º parâmetro a própria janela e como 2º parâmetro a *string* "Error:" seguida da mensagem que especifica o erro.



Interface usada na receção de eventos de ação:

```
public interface ActionListener {
  void actionPerformed(ActionEvent e);
}
```

Para a instanciação das caixas e da área de texto use os métodos:

```
public staticJTextField newJTextField( String title, int dim) {
  JTextField tf = new JTextField( dim );
  tf.setBorder( new TitledBorder( title ) ); return tf;
}
public static JTextArea newJTextArea( String title) {
  JTextArea ta = new JTextArea( 15, 35 );
  ta.setBorder( new TitledBorder( title ) ); return ta;
}
```