```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Diagnostics;
7  using System.IO;
8
9  namespace CodespaceJobCentre;
10
11 public static class Interface// Classe estática para definir a interface visual (cores e estilos).
12 {
13     public static string Bold => "\x1b[1m"; // Texto em negrito.
14     public static string Red => "\x1b[31m"; // Texto vermelho.
15     public static string Green => "\x1b[32m"; // Texto verde.
16     public static string Yellow => "\x1b[33m"; // Texto amarelo.
17     public static string OpcaoErro => $"\n{Interface.Red}Opção indisponível. {Interface.Reset}"; // Mensagem de erro.
18     public static string EscOp => $"{Interface.Bold}\nEscolha uma opção: {Interface.Reset} "; // Esquema do prompt para o utilizador.
19     public static string Reset => "\x1b[0m"; // Reset ao estilo padrão do texto.
20     public static void LimparTela() // Aguarda interação do utilizador.
21     {
22         Console.WriteLine("\nPressione qualquer tecla para continuar...");
23         Console.ReadKey();
24         Console.Clear(); // Limpa o ecrã.
25     }
26 }
27
28
29 public class Senha // uma senha para atendimento
30 {
31     public string Numero { get; set; } // Número da senha
32     public string TipoServico { get; set; } // Tipo de serviço associado à senha
33     public bool Atendida { get; set; } // Indica se a senha já foi atendida
34
35     public Senha(string numero, string tipoServico)
36     {
37         Numero = numero;
38         TipoServico = tipoServico;
39         Atendida = false; // Inicialmente, a senha não foi atendida
40     }
41
42     public void Atender() => Atendida = true; // Marca a senha como atendida
43 }
44
45 public class Servico
46 {
47     public string Nome { get; set; } // Nome do serviço.
48     public string Prefixo { get; set; } // Prefixo usado para gerar o número da senha
49     private Queue<Senha> filaSenhasPendentes = new Queue<Senha>(); // Queue para senhas Pendentes.
```

```
50      private List<Senha> filaSenhasAtendidas = new List<Senha>(); // Lista para senhas Atendidas.
51      private int contadorSenhas = 0; // Variável privada que será usada para contar o número em si da senha.
52
53      public Servico(string nome, string prefixo)
54      {
55          Nome = nome;
56          Prefixo = prefixo;
57      }
58
59      public void AtribuirSenha()
60      {
61          string numeroSenha;
62          contadorSenhas++; // Aumenta o contador de senha.
63          numeroSenha = $"{Prefixo}{contadorSenhas:D3}"; // Cria uma senha formatada com 3 espaços para números.
64          var senha = new Senha(numeroSenha, Nome);
65          Console.Clear();
66          filaSenhasPendentes.Enqueue(senha); // Adiciona a nova senha à fila.
67          Console.WriteLine($"\n{Interface.Yellow}Senha atribuída: {Interface.Bold}{senha.Numero} - {Nome}{Interface.Reset}"); // Confirmação da senha atribuída.
68      }
69
70      public Senha ChamarProximaSenha() // Método para chamar a próxima senha da fila e marcá-la como atendida.
71      {
72          if (filaSenhasPendentes.Count > 0) // Verifica se há senhas na fila.
73          {
74              Senha senha = filaSenhasPendentes.Dequeue(); // Retira senha da fila de senhas pendentes.
75              senha.Atender(); // Marca a senha como atendida.
76              filaSenhasAtendidas.Add(senha); // Adiciona à lista de senhas atendidas.
77              return senha;
78          }
79          return null;
80      }
81      public int QuantidadeSenhasPendentes() => filaSenhasPendentes.Count; // Método para retornar a quantidade de senhas que ainda estão pendentes.
82
83      public int QuantidadeSenhasAtendidas() => filaSenhasAtendidas.Count; // Método para retornar a quantidade de senhas que já foram atendidas.
84
85      public string UltimaSenhaAtendida()
86      {
87          if (filaSenhasAtendidas.Count > 0)
88              return filaSenhasAtendidas.Last().Numero; // Retorna o número da última senha atendida.
89          return "Nenhuma"; // Caso não haja senhas, retorna "Nenhuma".
90      }
91
92      public string UltimaSenha()
93      {
94          if (filaSenhasPendentes.Count > 0)
```

```
95              return filaSenhasPendentes.Last().Numero; // Retorna o número da
                  última senha gerada.
96          return "Nenhuma"; // Caso não haja senhas, retorna "Nenhuma".
97      }
98
99      public void ExibirSenhaPendentes() // Exibe o número de cada senha
          pendente.
100     {
101         foreach (var senha in filaSenhasPendentes) // Itera pela fila e
              demonstra as senhas pendentes.
102         {
103             Console.WriteLine($"- {senha.Numero}");
104         }
105     }
106 }
107
108 public class GestorDeFilas
109 {
110     private List<Servico> servicos; // Lista com os tipos de serviços.
111
112     public GestorDeFilas()
113     {
114         servicos = new List<Servico> // fácil possibilidade de acrésimo de
              serviços no futuro.
115         {
116             new Servico("Inscrição/Atualização de Dados", "A"), // 0.
117             new Servico("Apoio à Procura de Emprego", "B"), // 1.
118             new Servico("Informações Gerais", "C") // 2.
119         };
120     }
121
122     public void AtribuirSenha()
123     {
124         int escolha; // Variável da opcão que será escolhida pela utilizador.
125         Console.Clear();
126         MostrarMenuSecundário(); // Autoexplicativo.
127
128         if (int.TryParse(Console.ReadLine(), out escolha) && escolha > 0 &&
              escolha <= servicos.Count) // Lê a escolha do utilizador e verifica
              se é válida.
129         {
130             servicos[escolha - 1].AtribuirSenha();
131             Interface.LimparTela();
132         }
133         else if (escolha == 0) return; // Se for 0, retorna ao menu principal.
134
135         else // se for inválido, limpa o ecrã e mostra msg de erro.
136         {
137             Console.Clear();
138             Console.WriteLine(Interface.OpcaoErro); // Demostra erro.
139             Console.WriteLine("\nPressione qualquer tecla para
                  continuar..."); // Para o utilizador conseguir ler o console com
                   tempo.
140             Console.ReadKey();
141         }
142     }
```

```
143
144     public void ChamarProximaSenha()
145     {
146         int escolha; // Variável da opção que será escolhida pela utilizador.
147         Console.Clear();
148         MostrarMenuSecundário(); // Autoexplicativo.
149
150         if (int.TryParse(Console.ReadLine(), out escolha) && escolha > 0 &&
              escolha <= servicos.Count)  // Lê a escolha do utilizador e verifica
               se é válida.
151         {
152             Senha senha = servicos[escolha - 1].ChamarProximaSenha(); // i - 1
                   devido aos indices da lista.
153             if (senha != null)
154             {
155                 Console.Clear();
156                 Console.WriteLine($"\n{Interface.Green}Atendido:
                      {senha.Numero} - {senha.TipoServico}{Interface.Reset}");
157                 Interface.LimparTela();
158             }
159             else
160             {
161                 Console.Clear();
162                 Console.WriteLine($"\n{Interface.Red}Não há senhas pendentes
                      neste serviço.{Interface.Reset}");
163             }
164         }
165         else if (escolha == 0) return;  // Se não for válida, e se for 0,
              volta ao menu principal.
166
167         else // Se não for válida, nem for 0, exibe messagem de erro.
168         {
169             Console.Clear();
170             Console.WriteLine(Interface.OpcaoErro); // Demostra erro.
171             Console.WriteLine("\nPressione qualquer tecla para
                  continuar..."); // Para o utilizador conseguir ler o console com
                   tempo.
172             Console.ReadKey();
173         }
174
175     }
176     public void MostrarMenuSecundário() // Exibe o menu dos serviços.
177     {
178         Console.WriteLine($"\n{Interface.Bold}--- Selecione o serviço ---
              {Interface.Reset}\n");
179         for (int i = 0; i < servicos.Count; i++) // Exibe os serviços
              disponíveis com um loop for.
180         {
181             Console.WriteLine($"{Interface.Yellow}{i + 1}.{Interface.Reset}
                  {servicos[i].Nome}"); // i + 1 devido aos indices da lista.
182         }
183         Console.WriteLine($"{Interface.Red}0. Sair{Interface.Reset}");
184         Console.Write(Interface.EscOp); // Mostra no console, perguntando qual
               será a opção.
185     }
186
```

```
187     public void ConsultarEstatisticas() // Consulta local da execução do programa.
188     {
189         int totalPendentes = 0;
190         int totalAtendidas = 0;
191         string ultimaSenhaPendente = "Nenhuma";
192         string ultimaSenhaAtendida = "Nenhuma";
193         Console.Clear();
194         Console.WriteLine($"\n{Interface.Bold}--- Estatísticas ---{Interface.Reset}\n");
195         foreach (var tipoServico in servicos) // Itinera e mostra no console, o mesmo layout para cada serviço.
196         {
197             Console.WriteLine($"{Interface.Bold}{tipoServico.Nome}:{Interface.Reset}");
198             Console.WriteLine($"- Senhas atendidas: {tipoServico.QuantidadeSenhasAtendidas()}");
199             Console.WriteLine($"- Senhas pendentes: {tipoServico.QuantidadeSenhasPendentes()}");
200             Console.WriteLine($"- Última senha solicitada: {tipoServico.UltimaSenha()}");
201             Console.WriteLine($"- Última senha atendida: {tipoServico.UltimaSenhaAtendida()}\n");
202 
203             totalPendentes += tipoServico.QuantidadeSenhasPendentes();
204             totalAtendidas += tipoServico.QuantidadeSenhasAtendidas();
205 
206             if (tipoServico.QuantidadeSenhasPendentes() > 0) // Atualiza a última senha pendente (se houver)
207                 ultimaSenhaPendente = tipoServico.UltimaSenha();
208 
209             if (tipoServico.QuantidadeSenhasAtendidas() > 0) // Atualiza a última senha pendente (se houver)
210                 ultimaSenhaAtendida = tipoServico.UltimaSenhaAtendida();
211         }
212 
213         Console.WriteLine("\n--- Estatísticas Gerais ---\n"); // Demonstra as estatísticas gerais.
214         Console.WriteLine($"Total de senhas pendentes: {totalPendentes}");
215         Console.WriteLine($"Total de senhas atendidas: {totalAtendidas}");
216         Console.WriteLine($"Última senha pendente: {ultimaSenhaPendente}");
217         Console.WriteLine($"Última senha atendida: {ultimaSenhaAtendida}\n");
218     }
219 
220     public void MostrarMenuTerciário()
221     {
222         Console.WriteLine($"\n{Interface.Bold}--- Estatísticas ---{Interface.Reset}\n");
223         Console.WriteLine($"{Interface.Yellow}1.{Interface.Reset} Consultar");
224         Console.WriteLine($"{Interface.Yellow}2.{Interface.Reset} Exportar");
225         Console.WriteLine($"{Interface.Yellow}3.{Interface.Reset} Consultar e Exportar");
226         Console.WriteLine($"{Interface.Red}0. Sair{Interface.Reset}");
227         Console.Write(Interface.EscOp); // Mostra no console, perguntando qual será a opção.
228     }
```

```
229
230     public void ExportarEstatisticas(string caminhoArquivo) //
          Autoexplicativo.
231     {
232         try // Facilidade de identificar erros de exportação.
233         {
234             int totalPendentes = 0;
235             int totalAtendidas = 0;
236             string ultimaSenhaPendente = "Nenhuma";
237             string ultimaSenhaAtendida = "Nenhuma";
238             List<string> linhas = new List<string>();
239             foreach (var tipoServico in servicos) // Escreve o mesmo layout
                  para cada serviço.
240             {
241                 linhas.Add($"Serviço: {tipoServico.Nome}");
242                 linhas.Add($"Senhas atendidas:
                      {tipoServico.QuantidadeSenhasAtendidas()}");
243                 linhas.Add($"Senhas pendentes:
                      {tipoServico.QuantidadeSenhasPendentes()}");
244                 linhas.Add($"Última senha: {tipoServico.UltimaSenha()}");
245                 linhas.Add($"Última senha atendida:
                      {tipoServico.UltimaSenhaAtendida()}\n");
246                 totalPendentes += tipoServico.QuantidadeSenhasPendentes();
247                 totalAtendidas += tipoServico.QuantidadeSenhasAtendidas();
248             }
249
250             linhas.Add($"\n{Interface.Bold}--- Estatísticas Gerais ---
                  {Interface.Reset}\n"); // Demonstra as estatísticas gerais.
251             linhas.Add($"Total de senhas pendentes: {totalPendentes}"); //
                  Mostra o total de senhas pendentes no final da execução do
                  programa.
252             linhas.Add($"Total de senhas atendidas: {totalAtendidas}"); //
                  Mostra o total de senhas atendidas no final da execução do
                  programa.
253             linhas.Add($"Última senha pendente: {ultimaSenhaPendente}");
254             linhas.Add($"Última senha atendida: {ultimaSenhaAtendida}\n");
255
256
257             File.WriteAllLines(caminhoArquivo, linhas); // Escreve as linhas
                  no arquivo estabelecido.
258             Console.WriteLine($"{Interface.Green}Estatísticas exportadas para
                  {caminhoArquivo}{Interface.Reset}"); // Confirmação doa
                  exportação.
259
260         }
261         catch (Exception ex)
262         {
263             Console.WriteLine($"{Interface.Red}Erro ao exportar estatísticas:
                  {ex.Message}{Interface.Reset}"); // Explicação do erro de
                  exportação.
264         }
265     }
266
267     public void ExportadorNome()
268     {
269         string nomeArquivo; // Variável que será usada para guardar o nome do
```

```
            arquivo de texto que será exportado.
270         Console.Clear();
271         Console.WriteLine($"{Interface.Bold}0....{Interface.Reset} para voltar ao menu"); // 0 volta ao menu.
272         Console.WriteLine($"{Interface.Bold}null.{Interface.Reset} para data atual"); // Null deixará o nome do arquivo com a data do momento.
273         Console.Write($"\n{Interface.Bold}Digite o nome do arquivo para exportar as estatísticas: {Interface.Reset}");
274         nomeArquivo = Console.ReadLine(); // Input do utilizador.
275         if (nomeArquivo == "0") return; // volta ao menu principal.
276
277         if (string.IsNullOrWhiteSpace(nomeArquivo)) // se for null ou algum espaço atoa, vai dar o nome do arquivo com a data.
278         {
279             nomeArquivo = DateTime.Now.ToString("dd-MM-yyyy_H-mm");
280         }
281         ExportarEstatisticas(nomeArquivo + ".txt"); // Exporta as estatísticas com o nome do arquivo mais .txt.
282     }
283
284     public void ExibirFila() // Autoexplicativo.
285     {
286         Console.Clear();
287         Console.WriteLine($"\n{Interface.Bold}--- Fila Completa ---{Interface.Reset}");
288
289         foreach (var servico in servicos) // itinera com um loop foreach a lista de "servicos".
290         {
291             Console.WriteLine($"\n{Interface.Bold}{servico.Nome}:{Interface.Reset}"); // Título do serviço.
292             servico.ExibirSenhaPendentes(); // Exibir as senhas de cada serviço.
293         }
294
295         Console.WriteLine("\nPressione qualquer tecla para continuar..."); // Para o utilizador conseguir ler o console com tempo.
296         Console.ReadKey();
297     }
298 }
299
300 class Program
301 {
302     static void Main()
303     {
304         var gestor = new GestorDeFilas(); // Instatiação da classe GestorDeFilas.
305         bool continuar = true; // Execução do programa.
306
307         while (continuar) // Ciclo principal do programa, onde o utilizador pode escolher o que fazer.
308         {
309             MostrarMenuPrincipal();
310             MenuEscolha(ref continuar, ref gestor);
311         }
312     }
```

```
313     static void MenuEscolha(ref bool continuar, ref GestorDeFilas gestor)
314     {
315
316         string op1, op2; // Declaração de duas váriaveis para opções.
317         DateTime currentDate = DateTime.Now; // Variável para armazenar a hora
               atual para fins de estátistica.
318         op1 = Console.ReadLine(); // Seleciona a opção com base no menu
              principal.
319         switch (op1)
320         {
321             case "1": // Cria uma senha para tal serviço.
322                 gestor.AtribuirSenha();
323                 break;
324             case "2": // Atende uma senha para tal serviço.
325                 gestor.ChamarProximaSsenha();
326                 break;
327             case "3": // Tudo sobre as estatísticas, como menu e chamadas dos
                  respectivos métodos.
328                 Console.Clear();
329                 gestor.MostrarMenuTerciário();
330                 op2 = Console.ReadLine();
331                 switch (op2)
332                 {
333                     case "1": // Consulta.
334                         gestor.ConsultarEstatisticas();
335                         Interface.LimparTela();
336                         break;
337                     case "2": // Exportação.
338                         gestor.ExportadorNome();
339                         Interface.LimparTela();
340                         break;
341                     case "3": // Consulta e exportação.
342                         gestor.ExportadorNome(); // Define o nome do arquivo
                            que será exportado.
343                         gestor.ConsultarEstatisticas();
344                         Interface.LimparTela();
345                         break;
346                     case "0":
347                         return; // Sai para menu.
348                     default:
349                         Console.WriteLine(Interface.OpcaoErro); // default
                            demonstra erro.
350                         Interface.LimparTela();
351                         break;
352                 }
353                 break;
354             case "4": // Exibe as senhas pendentes.
355                 gestor.ExibirFila();
356                 break;
357             case "0": // Sai do programa COM backup das estatísticas.
358                 gestor.ExportarEstatisticas("estatisticas-backup.txt");
359                 continuar = false;
360                 break;
361             case "X": // Sai do programa SEM backup das estatísticas.
362                 Console.Clear();
363                 Console.WriteLine($"\n{Interface.Bold}--- Saída ---");
```

```
364                 Console.WriteLine($"\n1{Interface.Reset} - Finaliza\n
                      {Interface.Bold}Qualquer outra tecla{Interface.Reset} -
                      volta ao menu");
365                 Console.Write($"\n{Interface.Bold}{Interface.Yellow}
                      Confirmação de saída {Interface.Red}sem backup:
                      {Interface.Reset}");
366                 op2 = Console.ReadLine();
367                 if (op2 == "1") continuar = false;
368                 break;
369             default:
370                 Console.Clear();
371                 Console.WriteLine(Interface.OpcaoErro); // default demonstra
                      erro.
372                 Interface.LimparTela();
373                 break;
374         }
375         if (continuar) Console.Clear();
376     }
377     static void MostrarMenuPrincipal() // Menu de maior utilização do
          programa.
378     {
379         Console.Clear();
380         Console.WriteLine($"\n{Interface.Bold}--- Centro de emprego ---
              {Interface.Reset}\n");
381         Console.WriteLine($"{Interface.Yellow}1.{Interface.Reset} Atribuir
              senha");
382         Console.WriteLine($"{Interface.Yellow}2.{Interface.Reset} Chamar
              próxima senha");
383         Console.WriteLine($"{Interface.Yellow}3.{Interface.Reset} Consultar
              estatísticas");
384         Console.WriteLine($"{Interface.Yellow}4.{Interface.Reset} Exibir
              fila");
385         Console.WriteLine($"{Interface.Red}0.{Interface.Reset} {Interface.Red}
              Sair{Interface.Reset}");
386         Console.WriteLine($"{Interface.Red}{Interface.Bold}X. Sair sem backup
              {Interface.Reset}");
387         Console.Write(Interface.EscOp); // Mostra no console, perguntando qual
               será a opção.
388     }
389 }
```