**Exercícios**

1. **Consumo de energia**

Neste exercício, você irá simular o consumo de energia elétrica de diferentes dispositivos em uma casa (como ar-condicionado, geladeira, computador, chuveiro, ferro de passar roupa, etc).

Cada dispositivo elétrico consumirá uma quantidade de energia sempre que for utilizado. O consumo de todos os dispositivos será somado para gerar o consumo total da casa.

Quando o consumo total ultrapassar um valor limite (ex: 1000 Watts), o sistema deverá disparar um evento chamado ConsumoElevado.

Esse evento será baseado em um delegate personalizado, que envia informações sobre:

* O nome do dispositivo que causou o excesso (o de maior consumo).
* O consumo total atual da casa.

Crie classe assinante (que recebem a notificação do evento) deverá tomar uma ação automática, como:

* Exibir um alerta no console.
* Simular o desligamento de alguns dispositivos (ex: ar-condicionado ou aquecedor).

Requisitos Técnicos:

1. Classe DispositivoEletrico
   * Propriedades: Nome, ConsumoPorUso (em Watts), Ativo (bool).
   * Método Usar(): adiciona o consumo ao total da casa.
2. Classe SistemaEnergia
   * Armazena a lista de dispositivos.
   * Controla o consumo total.
   * Contém o evento ConsumoElevado.
   * Define o limite de consumo (ex: 1000W).
3. Delegate
   * Crie um delegate void AlertaConsumoHandler(string nomeDispositivo, double consumoAtual).
4. Evento ConsumoElevado
   * Disparado quando o consumo total ultrapassa o limite.
   * Deve chamar os métodos ouvintes.
5. Assinantes
   * Um método exibe uma mensagem de alerta no console.
   * Outro simula o desligamento automático de dispositivos de alto consumo.
6. **Biblioteca**

Você vai criar um pequeno sistema para registrar livros de uma biblioteca. Esses livros serão armazenados em uma lista e, em seguida, salvos em um arquivo .json no disco.

Depois, o sistema deve ler esse arquivo JSON, converter os dados de volta para objetos em memória e exibir os livros no console.

Crie uma classe Livro com as seguintes propriedades:

* Titulo (string)
* Autor (string)
* Ano (int)

1. **Controle de Usuário**

Você irá desenvolver um sistema simples para gerenciar usuários de um aplicativo. O sistema deve permitir o cadastro de usuários e realizar algumas ações automáticas ao longo do processo.

Cada vez que um novo usuário for cadastrado:

* Um evento chamado UsuarioCadastrado deve ser disparado.
* Esse evento deve ser baseado em um delegate personalizado, que define a assinatura dos métodos de notificação (ex: enviar email, registrar log etc.).
* Os ouvintes do evento deverão executar ações diferentes, como:
  + Exibir uma mensagem no console.
  + Gravar uma entrada em um arquivo de log .txt com data e hora do cadastro.
* O sistema deve manter uma lista de usuários na memória.
* Deve permitir que o usuário consulte os dados utilizando LINQ:
  + Filtrar usuários maiores de idade.
  + Agrupar usuários por domínio do email.
  + Listar os 3 mais recentes cadastrados.
* A lista de usuários deve ser serializada em JSON e salva em um arquivo.
* O sistema também deve permitir desserializar esse arquivo JSON para carregar os dados ao iniciar.

**Requisitos:**

1. Classe Usuario
   * Propriedades: Nome, Email, Idade, DataCadastro.
2. Classe GerenciadorUsuarios
   * Contém lista de usuários.
   * Método CadastrarUsuario(Usuario usuario):
     + Adiciona o usuário à lista.
     + Dispara o evento UsuarioCadastrado.
3. Delegate e Evento
   * Crie um delegate void NotificacaoHandler(Usuario usuario).
   * O evento UsuarioCadastrado será do tipo NotificacaoHandler.
4. Ouvintes(assinante) do Evento
   * Um método escreve no console: "Novo usuário: {Nome}, cadastrado em {DataCadastro}".
   * Outro grava no arquivo log.txt a mesma mensagem.
5. **Sistema de Cadastro de Produtos**

Você irá criar um sistema para cadastrar produtos. Toda vez que um novo produto for adicionado:

* Um evento ProdutoAdicionado deve ser disparado.
* Um método deve reagir a esse evento e gerar uma linha de relatório gravada com StreamWriter.
* O sistema deve permitir salvar e carregar os produtos em formato XML.
* Com LINQ, permita buscar produtos por faixa de preço ou categoria.

**Requisitos:**

1. Classe Produto
   * Nome, Categoria, Preco, DataCadastro.
2. Classe CatalogoProdutos
   * Método AdicionarProduto(Produto p) que dispara o evento.
   * Lista de produtos.
   * Métodos para salvar/carregar XML com XmlSerializer.
3. Evento e Delegate
   * Delegate void ProdutoHandler(Produto p).
   * Evento ProdutoAdicionado.
4. Gravação de Relatórios
   * Método na classe assinante para escreve relatório no arquivo relatorio.txt usando Stream.
5. Consultas LINQ
   * Produtos com preço entre 50 e 200.
   * Agrupar por categoria.
6. **Sistema de Registro de Ocorrências**

Você irá criar um sistema que registra ocorrências de incidentes em um condomínio. Toda vez que uma ocorrência for registrada:

* O evento OcorrenciaRegistrada deve ser disparado.
* Ouvintes(assinantes) devem reagir de formas diferentes:
  + Um grava no log ocorrencias.txt.
  + Outro exibe no console.
  + Se a ocorrência for urgente, exibir " ALERTA URGENTE “ no console.
* As ocorrências devem ser salvas em formato JSON.
* O sistema deve permitir buscar ocorrências por tipo ou urgência com LINQ.

**Requisitos:**

1. Classe Ocorrencia
   * Tipo, Local, Data, Urgente (bool).
2. Classe CentralDeOcorrencias
   * Método Registrar(Ocorrencia o) que dispara o evento.
   * Lista de ocorrências.
   * Métodos para salvar/carregar em JSON.
3. Evento e Delegate
   * Delegate void OcorrenciaHandler(Ocorrencia o).
   * Evento OcorrenciaRegistrada.
4. Ouvintes(assinantes)
   * Um grava no arquivo.
   * Outro escreve alertas no console.
   * Condicional especial para Urgente igual a true.
5. Consultas LINQ
   * Filtrar ocorrências urgentes.
   * Agrupar por tipo.