# Questões Conceituais

1. B – Falso – Objetos possuem identidade única.
2. A – Polimorfismo
3. B – Falso – Padrões de projetos descrevem soluções genéricas para determinados problemas, porém, de forma geral não oferecem uma implementação pronta ou um “componente executável” que pode ser utilizado para todos os casos. Trata-se de uma abordagem mais abstrata.
4. E - 0 2 1 0 2 1 0 2 1
   1. Verdadeiro
   2. Falso
   3. Falso

# Questões Práticas

## Sistema de Gestão de Produtos

O sistema desenvolvido oferece recursos para controle de estoque de produtos.

Além das telas de listagem e cadastro de estoque, foram desenvolvidas também as telas para listagem e cadastro de produtos e empresas.

Na tela de estoque há a opção de escolher um arquivo para importar os dados. Esse arquivo deve ser uma planilha com extensão .xsl ou. xslm e os dados devem estar no formato do template especificado nos requisitos (não houve alterações no template).

Cada registro de histórico passa pela seguinte validação antes de ser persistido:

***estoque atual do produto + novo registro de entrada – novo registro de saída = novo estoque***

Se o novo registro não satisfazer essa condição ele não será persistido e um mensagem de erro será enviada.

O processo de importação, além de validar o estoque para cada registro, também valida a formatação dos dados e verifica se no arquivo há mais de um nome de produto para o mesmo código. Toda a importação é executada em uma transação atômica, portanto, qualquer erro de validação ou algum tipo de exceção abortará a importação e nenhuma informações será persistida.

## Criação do Banco de Dados

Primeiramente, atualize a **ConnectionString** no arquivo **appsettings.json**, apontando para o banco a ser criado.

Existem 3 possibilidades de criação do banco de dados.

**Migrations**

1. No diretório raiz do projeto **GestaoProdutos**, abra o CMD e execute o comando **dotnet ef database update.** A execução desse comando criará o banco de dados, tabelas e índices. Assegure-se que o usuário tenha permissão para criação de banco de dados e demais comandos DDL. Uma alternativa é executar o comando **update-database** no **Package Manager Console** no Visual Studio, no escopo do projeto **GestaoProduto**.

**Restore**

1. Restaure o backup **db.backup**, que está no diretório raiz do repositório no GitHub**.**

**Scripts**

1. Execute o script **scripts.sql**, que está no diretório raiz do repositório no GitHub.

## Instalação de Dependências no Front-End

Após realizar o clone do repositório, executar **npm install** na raiz do projeto **GestaoProdutosClient**.

# Estrutura da Aplicação

Foi criada uma Solution com os projetos **GestaoProdutos** (back-end), **GestaoProdutosClient** (fornt-end) e **GestaoProdutosTest** (Testes).

### GestaoProdutos (back-end)

* **Controllers**: expôe os *endpoints* da Web Api. Não possui regras de negócio, mas encaminha as requisições para a camada de serviços e retornar os dados (se houver) junto com o status HTTP de acordo com a operação realizada.
* **Model**: contém as classes que modelam os dados da aplicação e são persistidos no banco de dados.
* **Services:** camada lógica da aplicação; possui todas as regras de negócios.
* **Helpers:** classes auxiliares, compartilhadas por toda a aplicação.
* **Exceptions:** classes de exceção personalizadas.
* **Migrations:** diretório criado automaticamente pelo Entity Framework Migrations com o registro de todas as alterações na modelagem do banco de dados durante o desenvolvimento da aplicação.

(Obs.: na abordagem escolhida para o desenvolvimento desse projeto, optei por não escrever uma camada customizada de persistência, pois seria redundante, uma vez que o Entity Framework já abstrai as operações de acesso a dados.)

### GestaoProdutosClient (fornt-end)

* **css**: arquivos de estilos.
* **js**: arquivos JavaScript.
  + **httpApi**: encapsula as chamadas AJAX utilizadas por toda a aplicação. Importante, pois se houver alteração de URIs no Web Service, por exemplo, não é necessário refatorar todo o código.
* **views:** páginas HTML.
* **lib:** bibliotecas de terceiros.

(Obs: caso seja necessário alterar a URL que aponta para a Web Api, basta alterar o arquivo **/js/config.js**.)

## GestaoProdutosTest (Testes)

A classe escolhida para realizar os testes unitários foi a classe **EstoqueService**. Por isso, foi criada a classe **EstoqueServicTest** na raiz do projeto **GestaoProdutosTest** para implementar os testes.

# Publicação

Antes de publicar o projeto, copie todo o conteúdo do projeto **GestaoProdutosClient** para o diretório **wwwroot** do projeto **GestaoProdutos** e altere “apiUrl” no arquivo **/js/config.js** para apontar para **/api/**. Certifique-se também de que a **ConnectionString** esteja correta.