**Testes Junit**

**Total de testes – 22 divididos entre 12 classes**  
**Classes Model:**  
  
Nós realizamos os mesmos dois testes em todas as classes Model por dois motivos cruciais:  
  
Primeiramente, ao executar o teste do construtor padrão, verificamos se a criação de uma instância não nula da classe é bem-sucedida. O teste é simples, consistindo na criação de uma nova instância da classe utilizando o construtor padrão, seguida da utilização do método assertNotNull do JUnit para confirmar que a instância criada não é nula. Isso é essencial para garantir que o construtor da classe esteja funcionando conforme o esperado. Quando o construtor padrão está corretamente implementado na classe, o teste passa, indicando que a instância do objeto é criada sem problemas.

Em segundo lugar, ao realizar o teste dos métodos setters e getters, verificamos se esses métodos funcionam corretamente. A sequência de ações é a seguinte:

Criamos uma nova instância da classe.

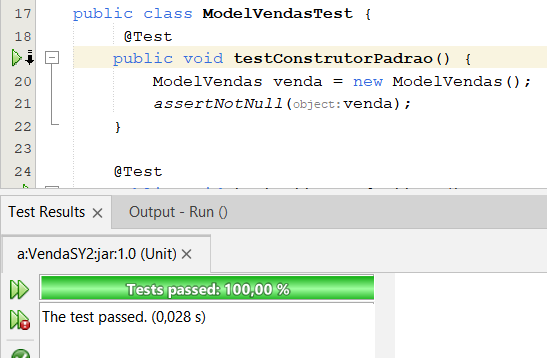
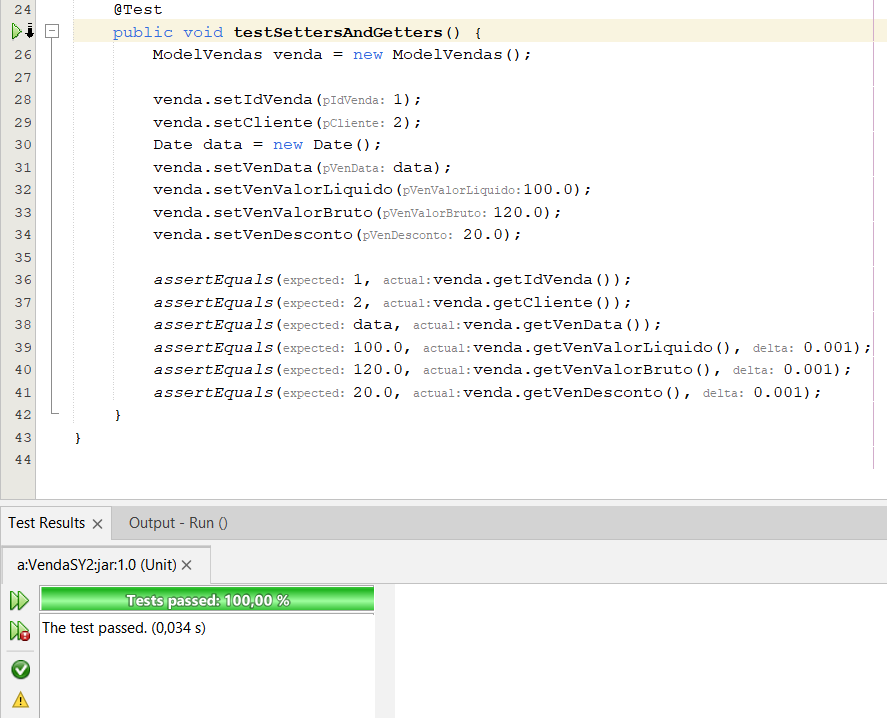
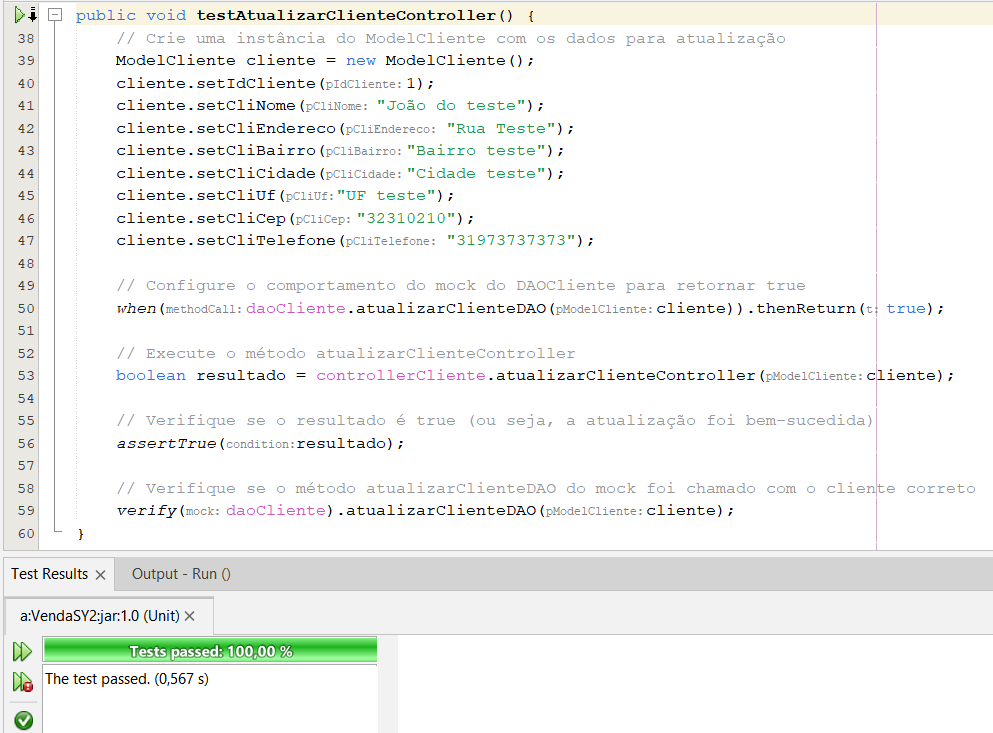
Utilizamos os métodos setters para definir os valores dos atributos do objeto.

Em seguida, utilizamos os métodos getters para obter os valores dos atributos do objeto.

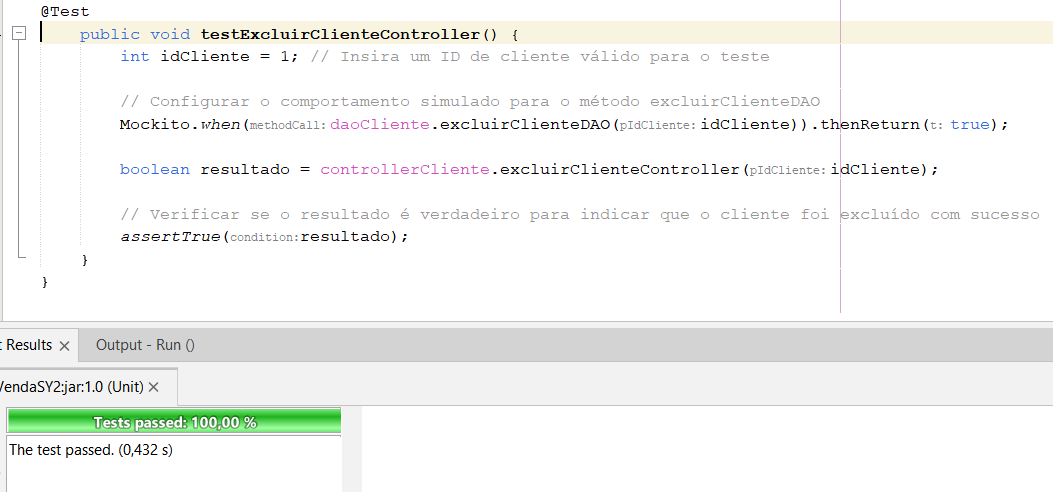
Usamos o método assertEquals do JUnit para comparar os valores obtidos pelos getters com os valores previamente definidos. Isso nos permite verificar se os métodos getters retornam os valores corretos.

Ao repetir esse teste em todas as classes do modelo, asseguramos que os métodos setters e getters de cada classe estejam devidamente implementados e funcionando adequadamente. Se os métodos estiverem corretos e os valores definidos nos setters corresponderem aos valores obtidos pelos getters, o teste passa sem lançar exceções. Esses testes são fundamentais para garantir que as classes do modelo estejam devidamente implementadas e que os atributos possam ser configurados e obtidos com precisão, independentemente da classe específica.

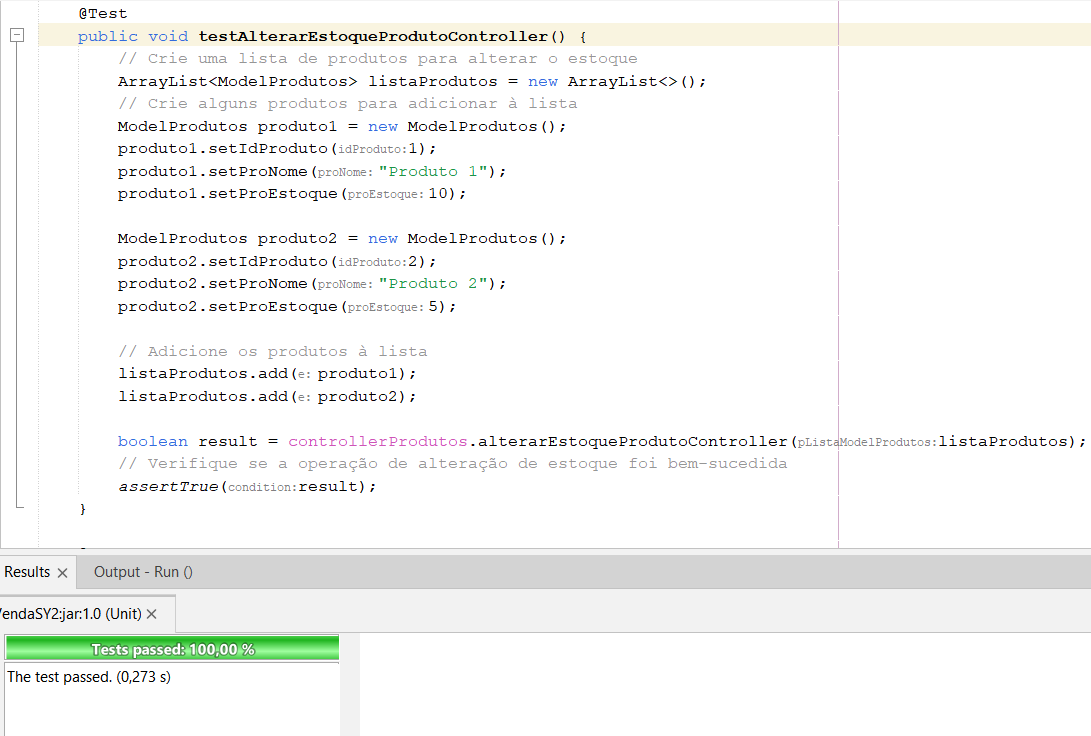
Foram realizados os 2 testes em 7 classes Model e todos passaram com 100%, seguem imagens da realização na classe ModelVendasTeste:

  
  
  
  
  
  
  
  
  
  
**Classes Controller:** Nas classes controller testamos diferentes tipos de métodos.  
  
**Classe ControllerClienteTest:**  
  
No primeiro teste, chamado testAtualizarClienteController, estamos focados em verificar a atualização de informações de um cliente. Para isso, criamos uma instância do ModelCliente com dados fictícios que seriam usados para atualizar o cliente. Em seguida, configuramos o comportamento simulado do DAOCliente (um objeto mock) para retornar true quando o método atualizarClienteDAO for chamado com o cliente criado. A ideia é simular uma operação bem-sucedida de atualização no banco de dados.  
  
Em seguida, chamamos o método atualizarClienteController da classe de controle e verificamos se o resultado é true, o que indica que a atualização foi bem-sucedida. Além disso, usamos o framework de teste Mockito para verificar se o método atualizarClienteDAO do objeto mock foi chamado com o cliente correto.  
  


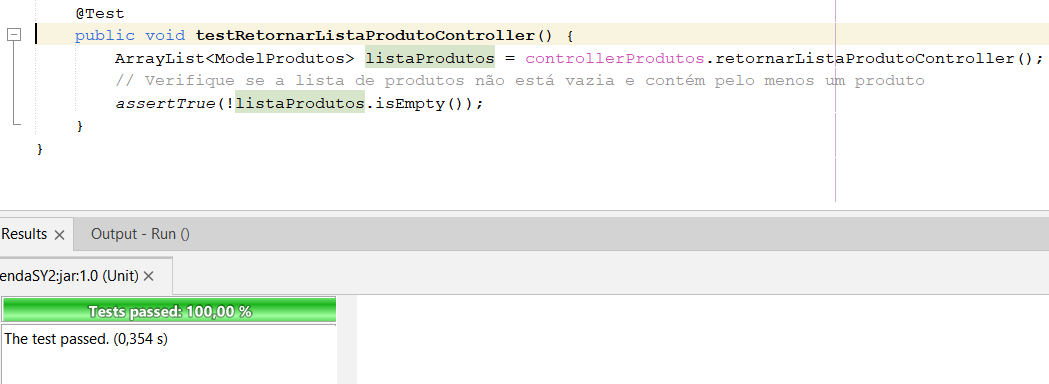
No segundo teste, chamado testExcluirClienteController, o foco está na exclusão de um cliente. Configuramos o comportamento simulado do método excluirClienteDAO do DAOCliente para retornar true quando chamado com um ID de cliente válido. Mais uma vez, a ideia é simular uma operação bem-sucedida de exclusão.

Chamamos o método excluirClienteController da classe de controle com um ID de cliente e verificamos se o resultado é true, indicando que a exclusão foi realizada com sucesso.  
  
  
  
A importância desses testes está em garantir que os métodos de atualização e exclusão de clientes funcionem corretamente. Eles verificam se o sistema está interagindo adequadamente com o banco de dados e se as operações de atualização e exclusão estão sendo executadas com êxito. Além disso, esses testes fornecem segurança de que o comportamento esperado é mantido ao longo do tempo, à medida que o código é modificado ou atualizado. Isso contribui para a estabilidade e confiabilidade do sistema.  
  
**Classe ControllerProdutosTest:**  
  
No primeiro teste, denominado testAlterarEstoqueProdutoController, nós estamos criando uma lista de produtos fictícia para simular uma operação de alteração de estoque. Essa lista inclui dois produtos de exemplo, com informações como ID, nome e estoque. A nossa intenção é verificar se a operação de alteração de estoque de produtos funciona corretamente.

Para isso, nós chamamos o método alterarEstoqueProdutoController da classe de controle e verificamos se o resultado retornado é true. Isso indica que a operação de alteração de estoque foi bem-sucedida. A importância desse teste está em garantir que a lógica de alteração de estoque esteja funcionando corretamente e que a quantidade de produtos seja atualizada conforme o esperado.

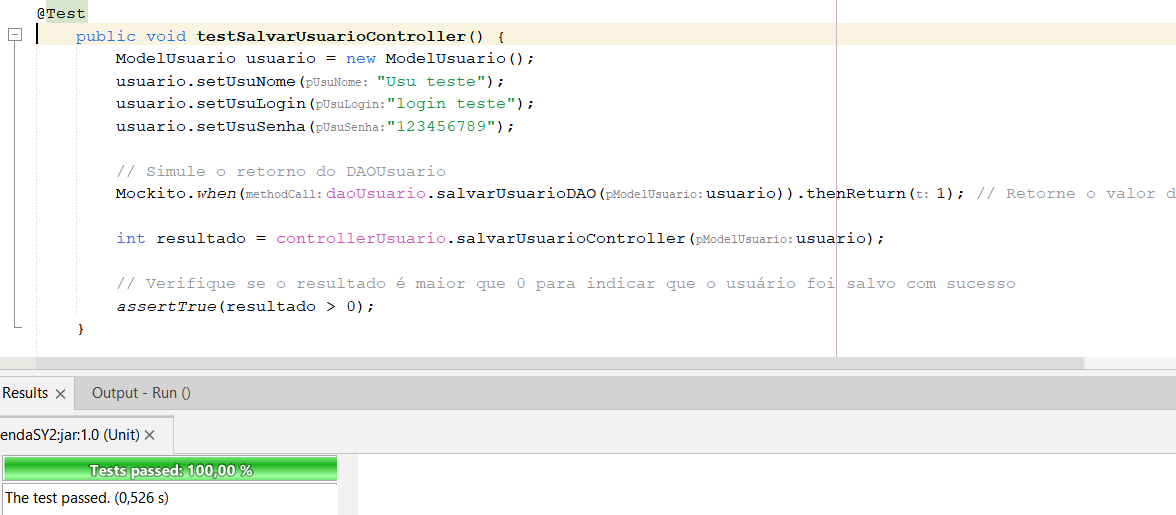
  
  
No segundo teste, chamado testRetornarListaProdutoController, estamos focados em verificar a funcionalidade de retorno da lista de produtos. Nós chamamos o método retornarListaProdutoController da classe de controle e verificamos se a lista de produtos não está vazia e contém pelo menos um produto.

A importância desse teste está em confirmar que o sistema pode recuperar com sucesso a lista de produtos disponíveis. Ele garante que a função de consulta esteja funcionando como esperado e que os produtos sejam recuperados corretamente, o que é crucial para a operação do sistema, pois a lista de produtos é frequentemente usada em várias partes da aplicação.

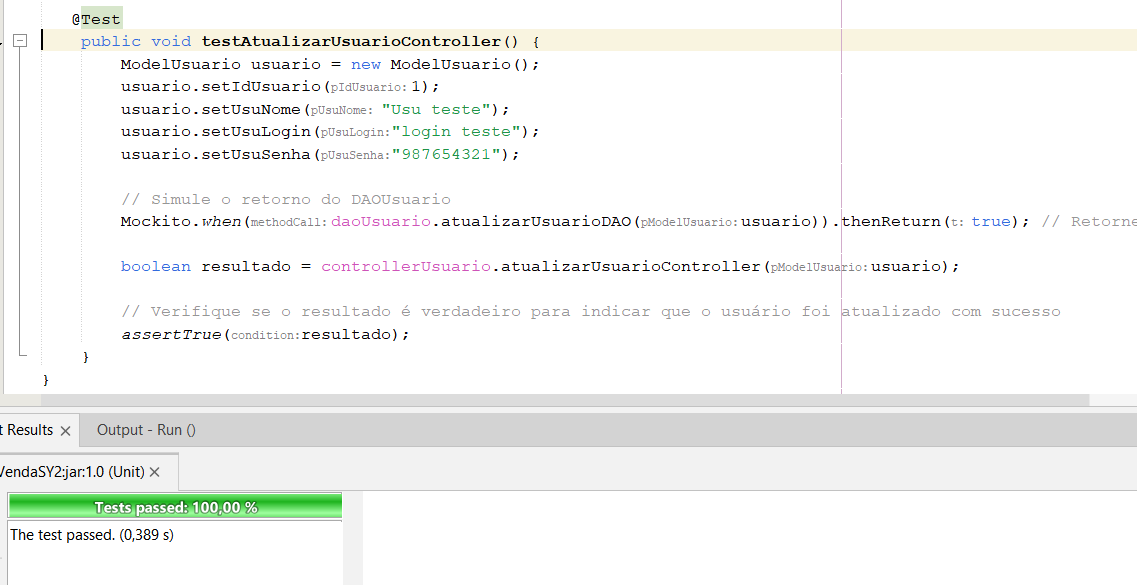
  
  
Em resumo, esses testes são fundamentais para garantir que a funcionalidade de alteração de estoque e recuperação da lista de produtos esteja implementada de maneira confiável e que o sistema possa manter a integridade de dados e operações eficazes em relação aos produtos gerenciados. Eles contribuem para a qualidade e robustez do sistema.  
  
  
  
  
  
**Classe ControllerUsuarioTeste:**  
  
Nesses testes, estamos avaliando a funcionalidade de salvar e atualizar um usuário em um sistema. Para isso, utilizamos a primeira pessoa do plural, pois estamos descrevendo como nós realizamos esses testes.

No primeiro teste, denominado testSalvarUsuarioController, nós criamos uma instância de um usuário com informações como nome, login e senha. Em seguida, simulamos o comportamento do DAO (Data Access Object) do usuário usando o framework Mockito. O DAO é responsável por interagir com o banco de dados para salvar o usuário. Neste caso, simulamos que a operação de salvar o usuário no banco de dados retornará um valor desejado (por exemplo, 1) quando chamada.

Após chamar o método salvarUsuarioController da classe de controle e obter o resultado, nós verificamos se o resultado é maior que 0. Essa verificação é crucial, pois um valor maior que 0 indica que a operação de salvar o usuário foi bem-sucedida. A importância desse teste é assegurar que o sistema seja capaz de adicionar novos usuários ao banco de dados com sucesso, garantindo a integridade dos dados.

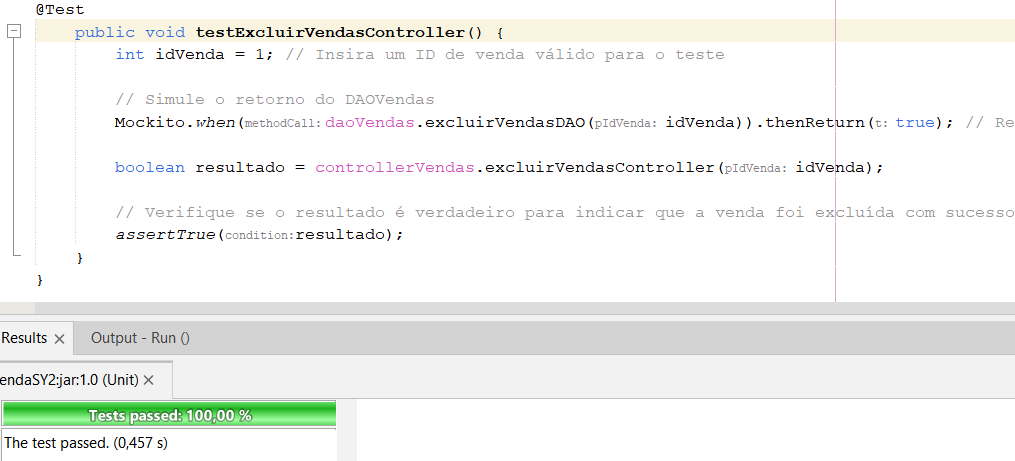
  
  
No segundo teste, chamado testAtualizarUsuarioController, nós criamos uma instância de usuário com informações de atualização, incluindo o ID do usuário, nome, login e senha. Novamente, utilizamos o Mockito para simular o comportamento do DAO de usuário, dessa vez simulando que a operação de atualização retorna true quando chamada.

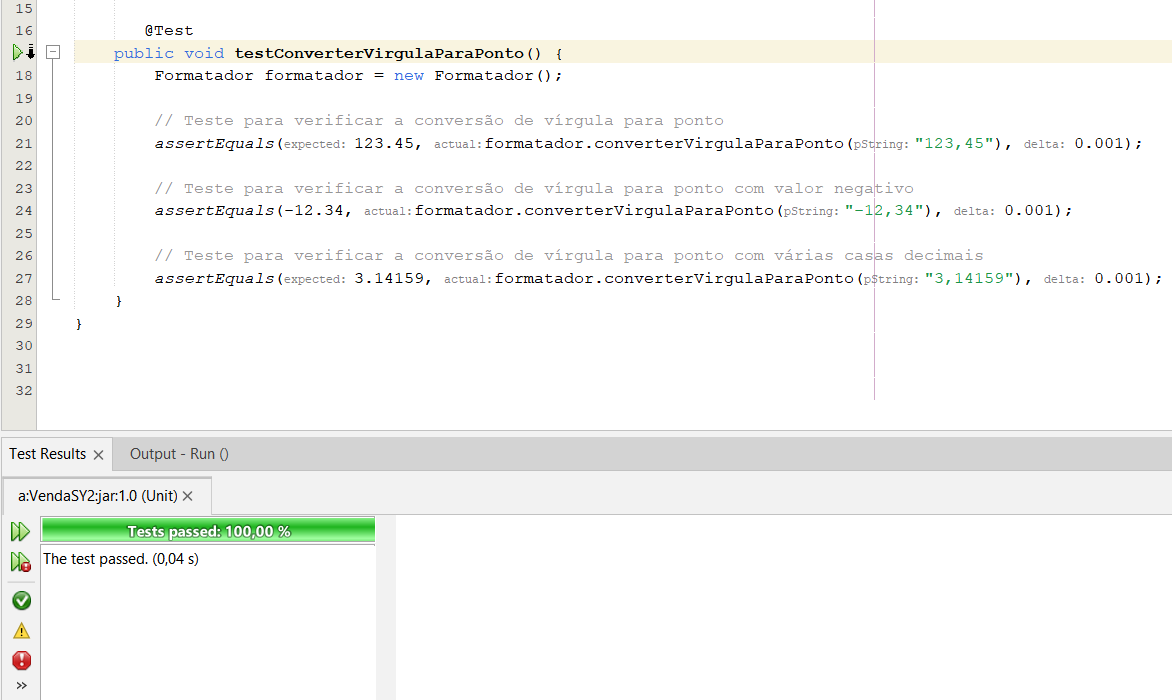
Após chamar o método atualizarUsuarioController e obter o resultado, verificamos se o resultado é verdadeiro. Isso indica que a operação de atualização de usuário foi bem-sucedida. A importância desse teste está em garantir que o sistema possa modificar as informações dos usuários no banco de dados de maneira confiável, o que é fundamental para manter os registros dos usuários atualizados e precisos.

  
  
**Classe ControllerVendasTest:**  
  
Nesse teste, estamos descrevendo o que aconteceu quando realizamos a avaliação da funcionalidade de exclusão de vendas em um sistema. Utilizamos a primeira pessoa do plural, pois estamos compartilhando como nós conduzimos o teste e sua importância.

Primeiramente, definimos um ID de venda válido (neste caso, 1) que será usado como entrada para o teste. Em seguida, usamos o framework Mockito para simular o comportamento do DAO de vendas. O DAO é responsável por interagir com o banco de dados para excluir uma venda. Simulamos que, quando chamamos a operação excluirVendasDAO do DAO, ela retornará true. Essa simulação permite que o teste avalie o comportamento do sistema sem a necessidade de acessar efetivamente o banco de dados.

Após configurar o comportamento do DAO, chamamos o método excluirVendasController da classe de controle com o ID de venda definido. O resultado da operação de exclusão é armazenado na variável resultado.

  
  
  
  
  
  
**Classe FormatadorTest:**  
  
  
Nesse teste, estamos verificando o método converterVirgulaParaPonto de uma classe chamada Formatador. A importância desse teste é verificar se o método está corretamente convertendo números no formato brasileiro, que usa vírgula como separador decimal, para o formato padrão que utiliza ponto como separador decimal. O método aceita uma string que representa um número com vírgula como separador decimal e deve retornar um valor de ponto flutuante correspondente.

Estamos verificando a conversão para diferentes cenários, incluindo números positivos, negativos e números com várias casas decimais. Através do uso do método assertEquals do framework de teste (presumivelmente JUnit), estamos comparando o valor retornado pelo método converterVirgulaParaPonto com o valor esperado.  
  
   
  
  
A importância desse teste reside em garantir que o método converterVirgulaParaPonto funcione corretamente, convertendo corretamente números no formato brasileiro para o formato padrão com ponto decimal. Isso é particularmente importante em sistemas que precisam lidar com entrada de dados de usuários que utilizam diferentes convenções para separadores decimais. Portanto, estamos verificando se o método de formatação está adequadamente implementado e se pode realizar com precisão a conversão necessária, evitando erros de interpretação de valores em diferentes formatos.