Desenvolvimento Full Stack

Felipe Marques de Almeida - 202208291929

Polo Rodovia Br 407 - Juazeiro - Ba
RPG0018 - Por que não paralelizar – Turma 22.3 – 3º Semestre
GitHub: Kifflom2108/Missao-Pratica-Nivel-5-Mundo-3 (github.com)

Objetivo da Prática

Criar servidores Java com base em Sockets.

Criar clientes síncronos para servidores com base em Sockets.

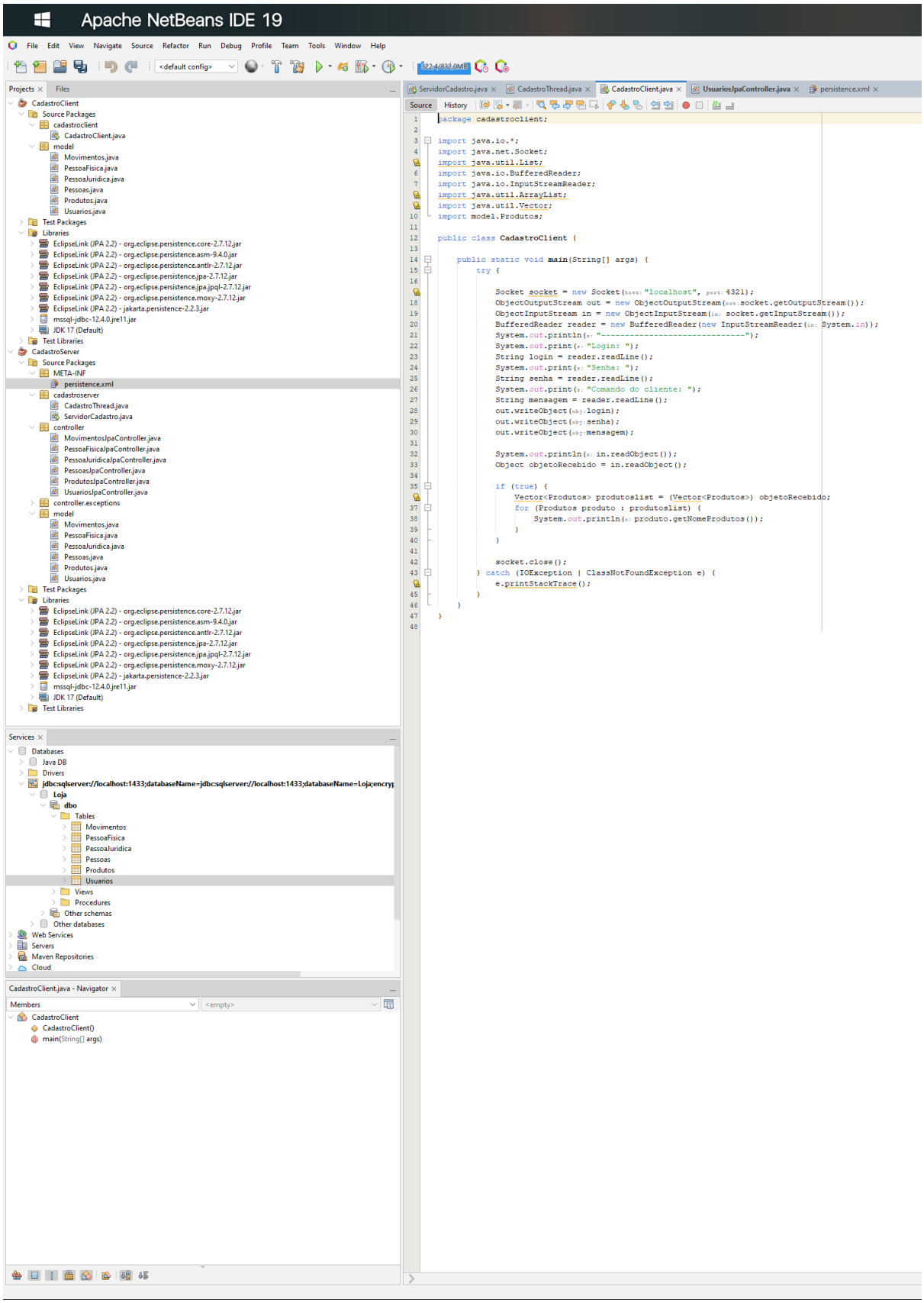
Criar clientes assíncronos para servidores com base em Sockets.

Utilizar Threads para implementação de processos paralelos.

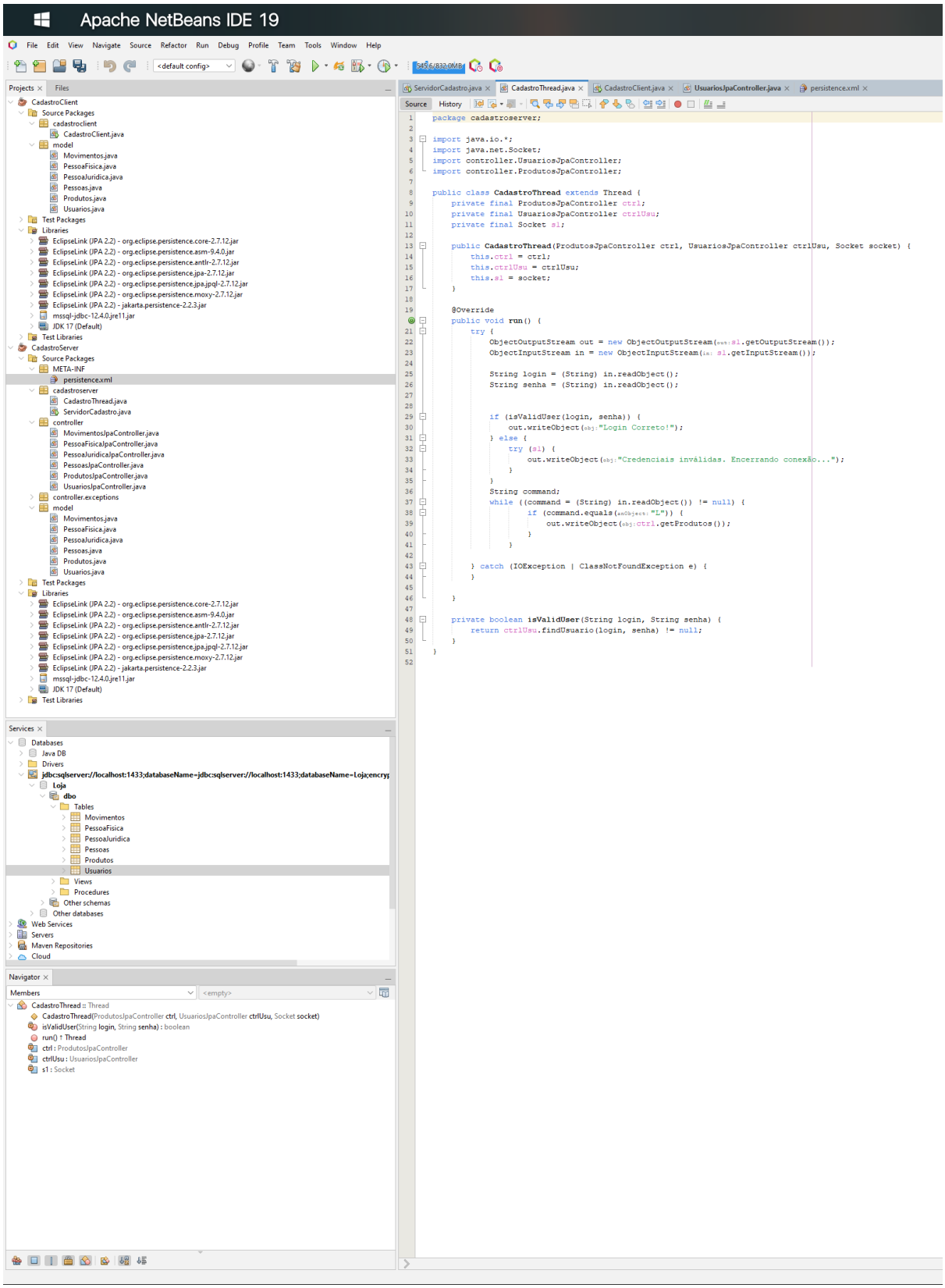
No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

Codigos: (Todas as imagens com qualidade original estão disponível no repositório!)

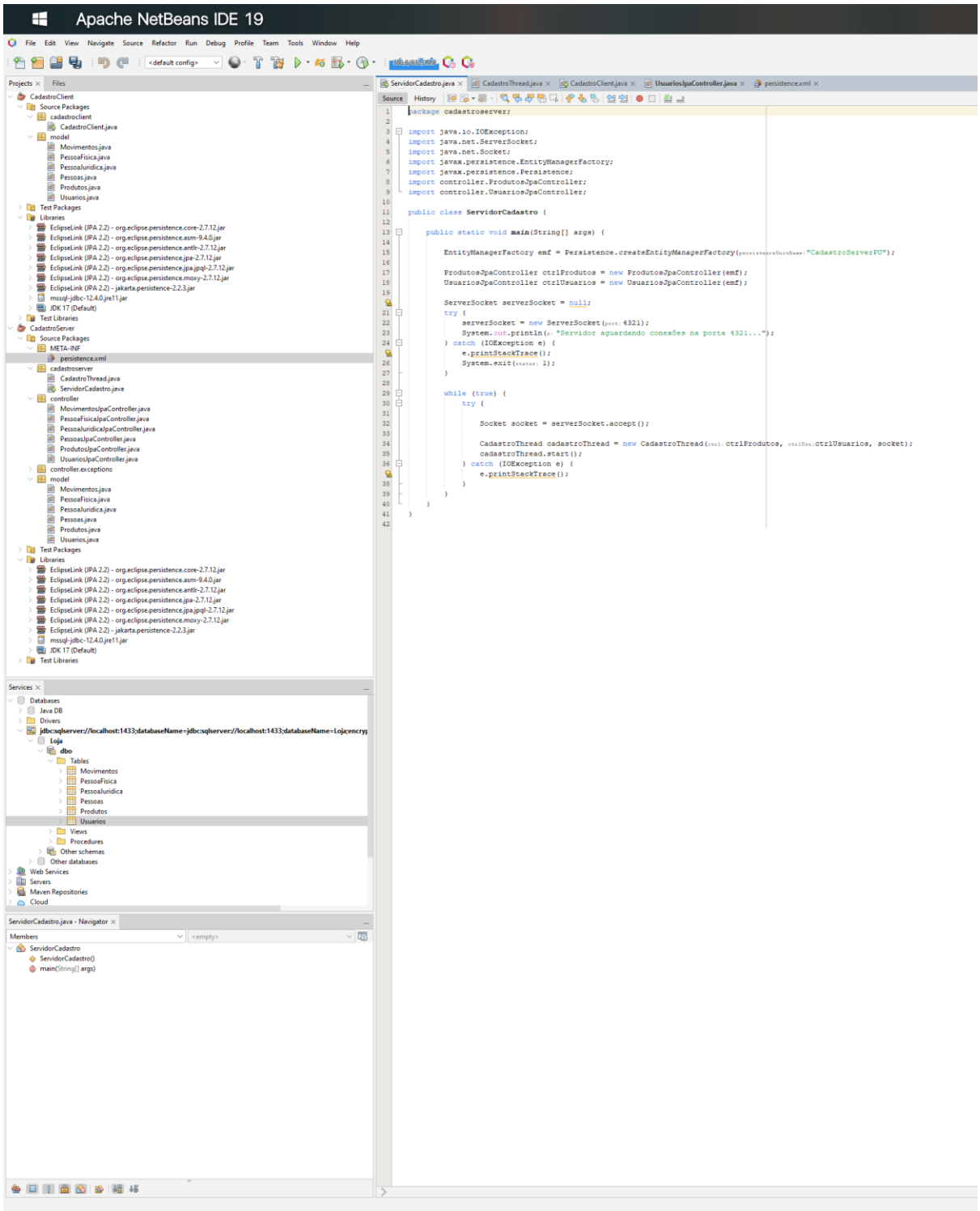
CadastroClient:



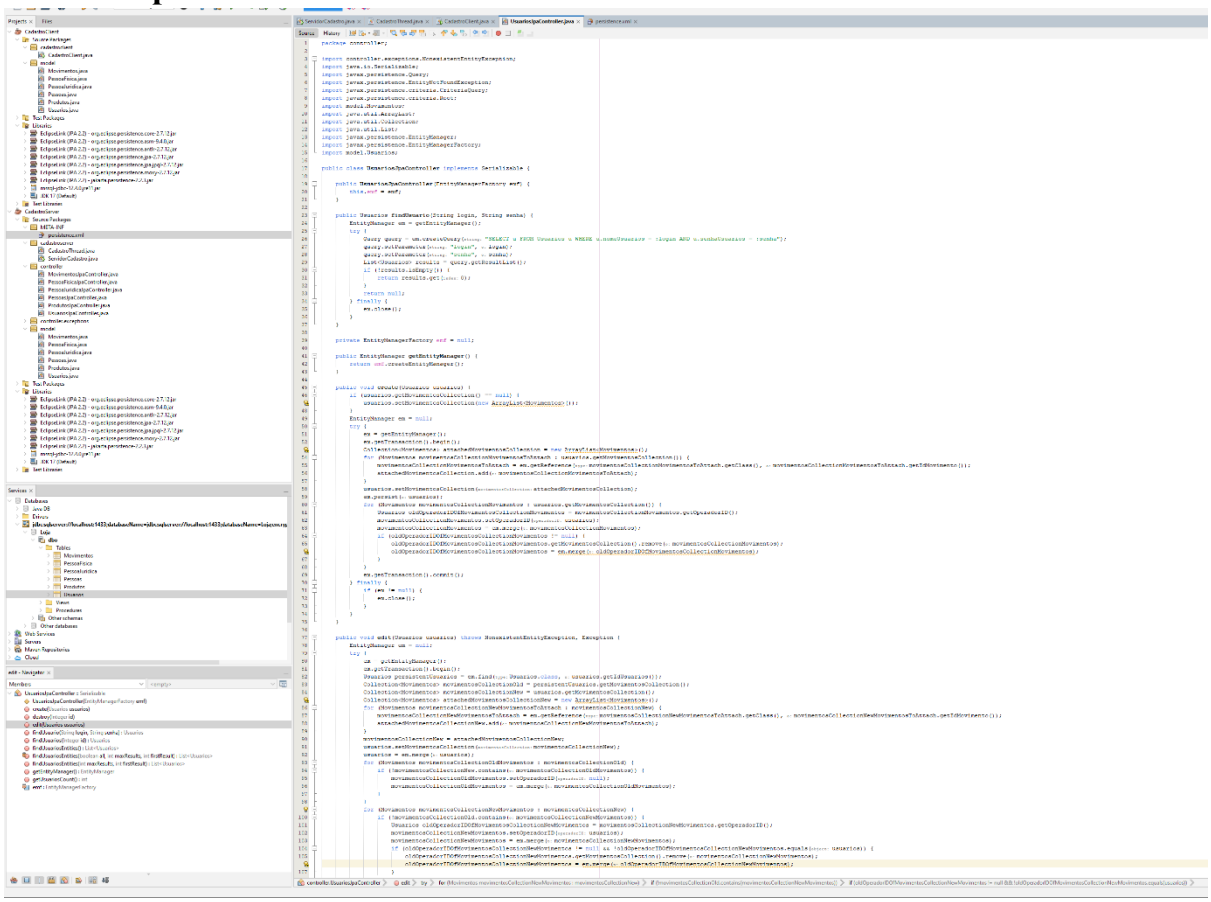
CadastroThread:

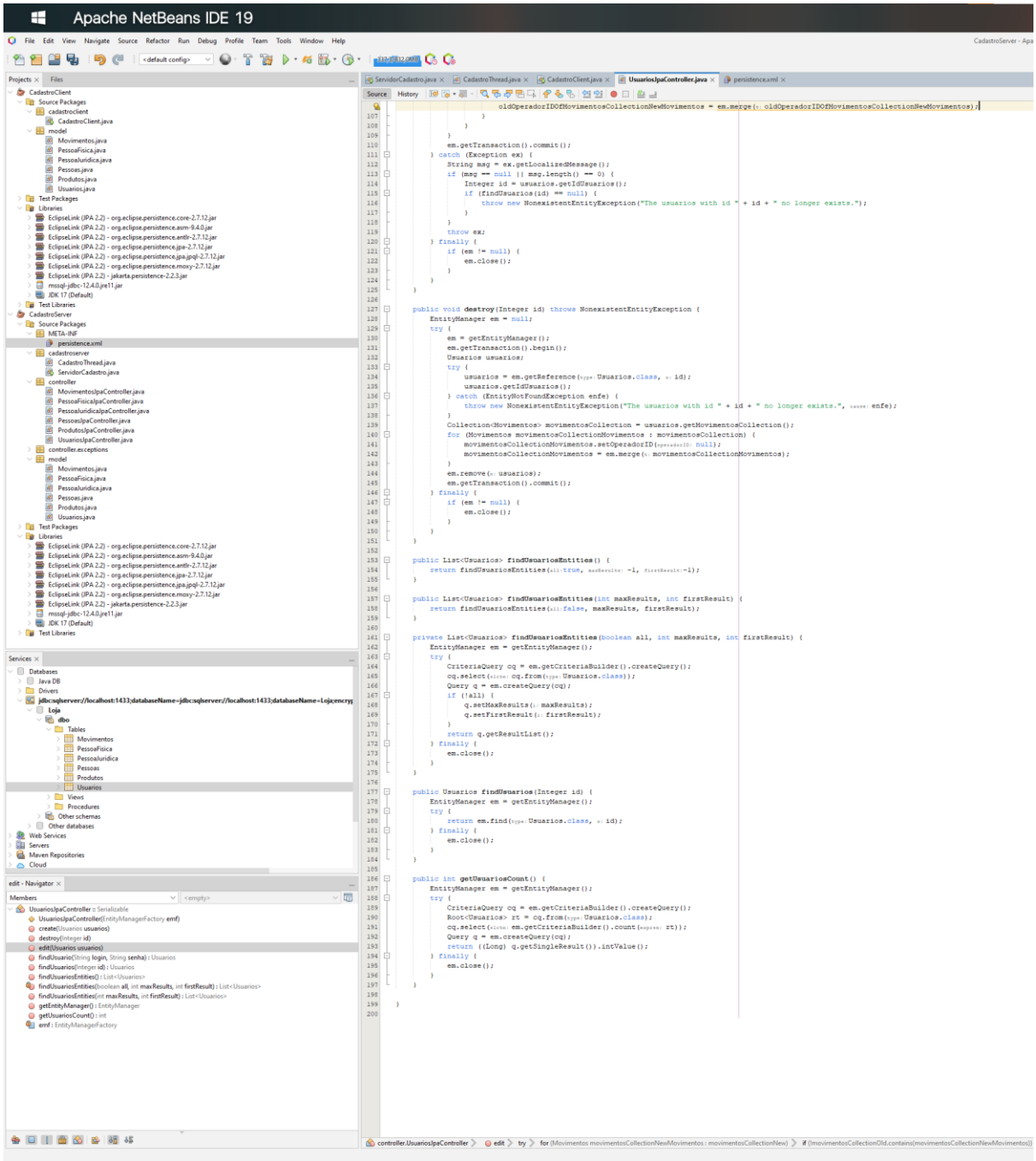


ServidorCadastro:

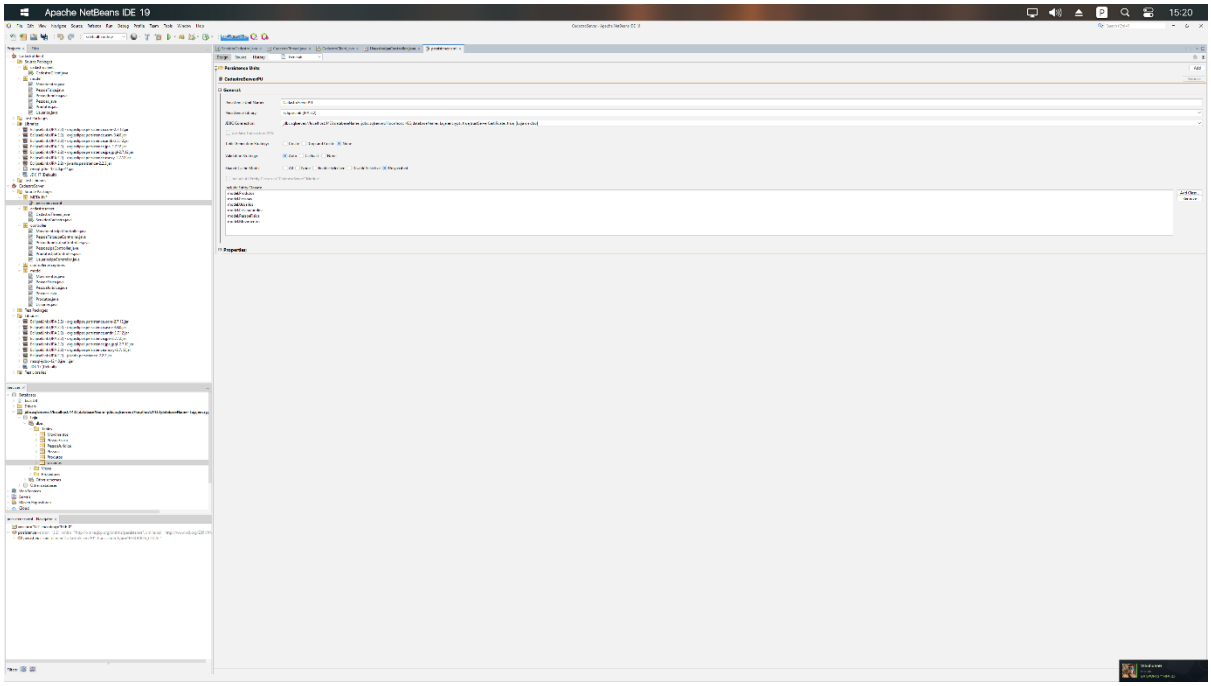


UsuariosJpaController:



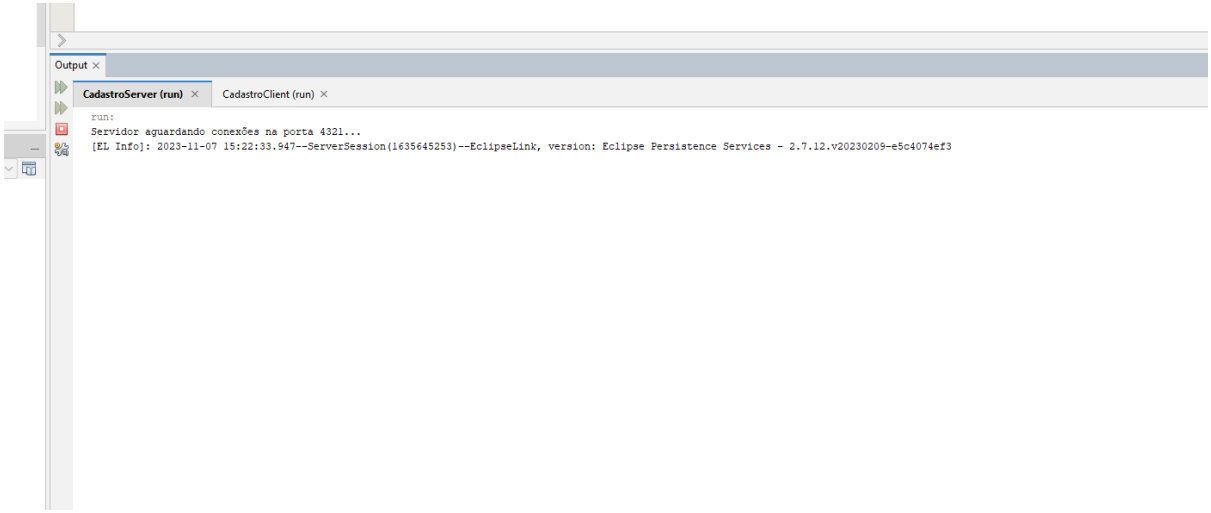

Apache NetBeans IDE 19
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
Files
CadastroClient
Source Packages
cadastroclient
CadastroClient.java
model
Movimentos.java
PessoaFisica.java
PessoaJuridica.java
Pessoas.java
Produtos.java
Usuarios.java
Test Packages
Libraries
EclipseLink (JPA 2.2) - org.eclipse.persistence.core-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.asm-9.4.0.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.antlr-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa.jpql-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.moxy-2.7.12.jar
EclipseLink (JPA 2.2) - jakarta.persistence-2.2.3.jar
mssql-jdbc-12.4.0.jre11.jar
JDK 17 (Default)
Test Libraries
CadastroServer
Source Packages
META-INF
persistence.xml
cadastroserver
CadastroThread.java
ServidorCadastro.java
controller
MovimentosJpaController.java
PessoaFisicaJpaController.java
PessoaJuridicaJpaController.java
PessoasJpaController.java
ProdutosJpaController.java
UsuariosJpaController.java
controller.exceptions
model
Movimentos.java
PessoaFisica.java
PessoaJuridica.java
Pessoas.java
Produtos.java
Usuarios.java
Test Packages
Libraries
EclipseLink (JPA 2.2) - org.eclipse.persistence.core-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.asm-9.4.0.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.antlr-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa.jpql-2.7.12.jar
EclipseLink (JPA 2.2) - org.eclipse.persistence.moxy-2.7.12.jar
EclipseLink (JPA 2.2) - jakarta.persistence-2.2.3.jar
mssql-jdbc-12.4.0.jre11.jar
JDK 17 (Default)
Test Libraries
Services
Databases
Java DB
Drivers
jdbc:sqlserver://localhost:1433;databaseName=jdbc:sqlserver://localhost:1433;databaseName=Loja;encryp
Loja
dbo
Tables
Movimentos
PessoaFisica
PessoaJuridica
Pessoas
Produtos
Usuarios
Views
Procedures
Other schemas
Other databases
Web Services
Servers
Maven Repositories
Cloud
edit - Navigator
Members
UsuariosJpaController :: Serializable
UsuariosJpaController(EntityManagerFactory emf)
create(Usuarios usuarios)
destroy(Integer id)
edit(Usuarios usuarios)
findUsuario(String login, String senha) : Usuarios
findUsuarios(Integer id) : Usuarios
findUsuariosEntities() : List<Usuarios>
findUsuariosEntities(boolean all, int maxResults, int firstResult) : List<Usuarios>
findUsuariosEntities(int maxResults, int firstResult) : List<Usuarios>
getEntityManager() : EntityManager
getUsuariosCount() : int
emf : EntityManagerFactory
ServidorCadastro.java
CadastroThread.java
CadastroClient.java
UsuariosJpaController.java
persistence.xml
Source
History
oldOperadorIDOfMovimentosCollectionNewMovimentos = em.merge(oldOperadorIDOfMovimentosCollectionNewMovimentos);
} catch (Exception ex) {
String msg = ex.getLocalizedMessage();
if (msg == null || msg.length() == 0) {
Integer id = usuarios.getIdUsuarios();
if (findUsuarios(id) == null) {
throw new NonexistentEntityException("The usuarios with id " + id + " no longer exists.");
throw ex;
} finally {
if (em != null) {
em.close();
public void destroy(Integer id) throws NonexistentEntityException {
EntityManager em = null;
em = getEntityManager();
em.getTransaction().begin();
Usuarios usuarios;
usuarios = em.getReference(Usuarios.class, id);
usuarios.getIdUsuarios();
} catch (EntityNotFoundException enfe) {
throw new NonexistentEntityException("The usuarios with id " + id + " no longer exists.", enfe);
Collection<Movimentos> movimentosCollection = usuarios.getMovimentosCollection();
for (Movimentos movimentosCollectionMovimentos : movimentosCollection) {
movimentosCollectionMovimentos.setOperadorID(null);
movimentosCollectionMovimentos = em.merge(movimentosCollectionMovimentos);
em.remove(usuarios);
em.getTransaction().commit();
public List<Usuarios> findUsuariosEntities() {
return findUsuariosEntities(true, -1, -1);
public List<Usuarios> findUsuariosEntities(int maxResults, int firstResult) {
return findUsuariosEntities(false, maxResults, firstResult);
private List<Usuarios> findUsuariosEntities(boolean all, int maxResults, int firstResult) {
EntityManager em = getEntityManager();
try {
CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
cq.select(cq.from(Usuarios.class));
Query q = em.createQuery(cq);
if (!all) {
q.setMaxResults(maxResults);
q.setFirstResult(firstResult);
return q.getResultList();
em.close();
public Usuarios findUsuarios(Integer id) {
EntityManager em = getEntityManager();
try {
return em.find(Usuarios.class, id);
em.close();
public int getUsuariosCount() {
EntityManager em = getEntityManager();
try {
CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
Root<Usuarios> rt = cq.from(Usuarios.class);
cq.select(em.getCriteriaBuilder().count(rt));
Query q = em.createQuery(cq);
return ((Long) q.getSingleResult()).intValue();
em.close();
}
controller.UsuariosJpaController edit try for (Movimentos movimentosCollectionNewMovimentos : movimentosCollectionNew) if (!movimentosCollectionOld.contains(movimentosCollectionNewMovimentos))

Persistence.xml:

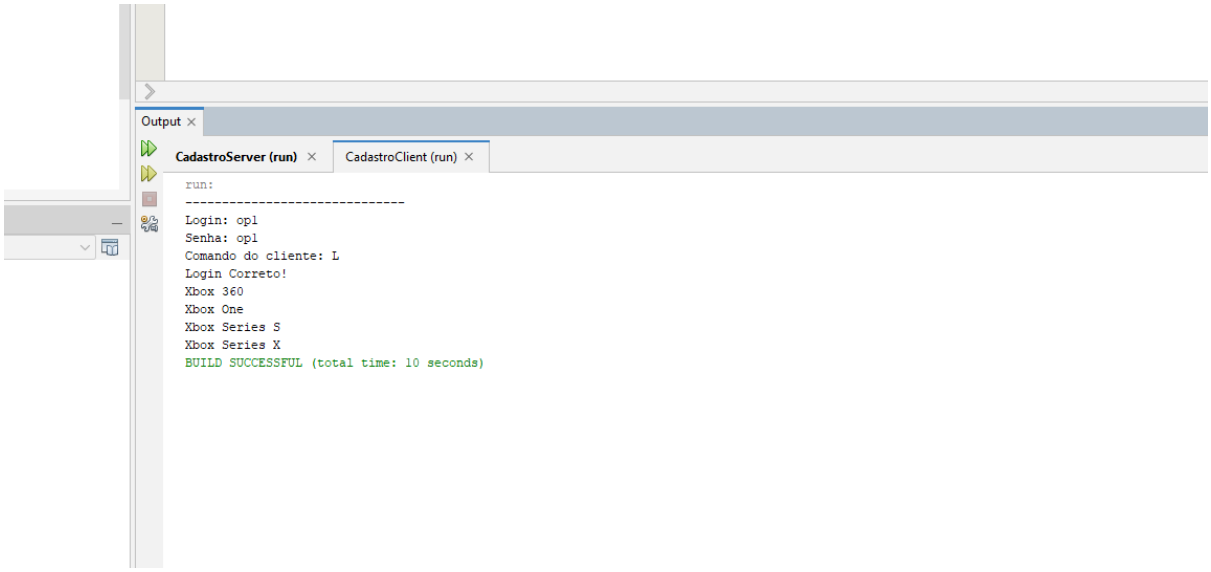


Resultados:

RunServidor:



RunClient:



Análise e Conclusão:

As classes Socket e ServerSocket desempenham papéis fundamentais na comunicação entre sistemas em uma rede. O Socket é usado para criar um ponto de comunicação em um determinado dispositivo, enquanto o ServerSocket age como o "ouvinte" que aguarda e aceita solicitações de conexão. Quando um cliente deseja se comunicar com um servidor, ele inicia uma conexão com o ServerSocket, que permite a transferência de dados entre as partes.

As portas desempenham um papel crucial na conexão com servidores, pois elas fornecem um meio de direcionar o tráfego para os aplicativos corretos em um servidor. Cada aplicativo que aceita conexões de rede geralmente usa uma porta específica para receber e processar dados. Portanto, as portas garantem que os dados sejam direcionados ao aplicativo correto no servidor.

As classes de entrada e saída ObjectInputStream e ObjectOutputStream são usadas para transmitir objetos entre sistemas. Elas são importantes porque permitem a comunicação de dados complexos em formato de objeto. Além disso, a serialização é necessária para que os objetos sejam transmitidos de forma eficaz e reconstruídos com precisão no destino. A serialização converte objetos em uma sequência de bytes que podem ser transmitidos pela rede e posteriormente desserializados para recriar o objeto original.

Ao utilizar as classes de entidades JPA no cliente, é possível garantir o isolamento do acesso ao banco de dados devido ao conceito de persistência de dados. O JPA (Java Persistence API) fornece uma camada de abstração sobre o banco de dados, permitindo que as operações de banco de dados sejam realizadas sem a necessidade de escrever SQL diretamente. Isso ajuda a manter o acesso aos dados consistente e protegido, evitando que o cliente acesse diretamente o banco de dados subjacente. Portanto, mesmo no cliente, o acesso aos dados é gerenciado de forma isolada e controlada, seguindo os princípios de persistência e abstração de banco de dados.