CURSO DESENVOLVIMENTO FULL STACK
POLO JACAREPAGUÁ - RIO DE JANEIRO – RJ

Disciplina: Nível 1: Iniciando o Caminho Pelo Java

Semestre: 2º - 2024

Aluno: GILBERTO FERREIRA DA SILVA JUNIOR

Matrícula: 2023.0701.4923

Repositório do Projeto: https://github.com/GilbertoFSJunior/CadastroPOO.git

(RPG0014---Iniciando-o-caminho-pelo-Java (github.com))

Relatório discente de acompanhamento

Título da Prática: RPG0014 - Iniciando o caminho pelo Java

Objetivo da Prática: Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java. Utilizar herança e polimorfismo na definição de entidades, utilizar persistência de objetos em arquivos binários, implementar uma interface cadastral em modo texto, utilizar o controle de exceções da plataforma Java.

Foram implementadas as classes:

- Pessoa;
- PessoaFisica;
- PessoaJuridica;
- PessoaFisicaRepo;
- PessoaJuridicaRepo;
- Main;

Classe Pessoa:

Construtor Padrão: Ele é útil quando você quer criar um objeto sem definir seus valores imediatamente. Isso é importante em várias situações, como ao carregar dados de um arquivo onde você define os valores posteriormente.

Construtor Completo: Esse permite que você crie uma Pessoa já com id e nome definidos.

Método exibir(): Ele imprime os valores dos atributos id e nome no console, facilitando a verificação dos dados.

Getters e Setters: Esses métodos são práticas comuns em Java para acessar (get) e modificar (set) os valores de atributos privados. Isso segue o princípio de encapsulamento, onde você mantém os atributos privados e controla o acesso a eles.

Classes PessoaFisica / PessoaJuridica:

Construtor Completo: Agora inclui o parâmetro idade, garantindo que todos os atributos sejam inicializados ao criar uma nova PessoaFisica.

Método exibir(): Adicionei a exibição da idade, assim como já estava sendo feito para o cpf. Isso garante que todas as informações relevantes da PessoaFisica sejam mostradas.

Interface Serializable: Não esqueça de implementar a interface Serializable para permitir a persistência em arquivos.

Anotação @Override: A anotação @Override foi adicionada ao método exibir(). Isso ajuda o compilador a identificar que este método está sobrescrevendo um método na classe base (Pessoa), melhorando a legibilidade e evitando erros.

Nome do Campo cnpj: Seguindo a convenção camelCase, usamos cnpj ao invés de CNPJ. Isso torna o código mais consistente com as práticas de codificação em Java.

Método Setter: Ajustei o nome do método setter para setCnpj, de modo a refletir corretamente o campo que ele está modificando.

Classe PessoaFisicaRepo / PessoaJuridicaRepo:

Campo listaPessoaFisica: Esse é o ArrayList que armazena todas as instâncias de PessoaFisica que você inserir no repositório.

Construtor: Inicializa a lista de PessoaFisica.

Método inserir(PessoaFisica pf): Adiciona uma nova pessoa física ao repositório.

Método alterar(PessoaFisica pf): Modifica uma pessoa física existente. Ele percorre a lista e substitui a pessoa física com o mesmo id pela nova versão que você passar como argumento.

Método excluir(int id): Remove uma pessoa física do repositório com base no id.

Método obter(int id): Retorna uma pessoa física com o id correspondente. Se não encontrar, retorna null.

Método obterTodos(): Retorna todas as pessoas físicas no repositório como uma lista.

Método persistir(String nomeArquivo): Salva o conteúdo do repositório em um arquivo no disco. Ele serializa o ArrayList para que possa ser recuperado mais tarde.

Método recuperar(String nomeArquivo): Lê o conteúdo de um arquivo no disco e recupera o repositório, desserializando o ArrayList.

Importante:

- **Tratamento de Exceções:** Nos métodos persistir e recuperar, estamos lançando (throws) as exceções IOException e ClassNotFoundException, que são comuns ao trabalhar com arquivos e serialização. Você precisará tratar essas exceções no código que chamar esses métodos.
- **Serialização:** Como todas as classes (Pessoa, PessoaFisica, etc.) implementam Serializable, o ArrayList que contém essas instâncias pode ser serializado e desserializado sem problemas.

Classe Main:

1. **Instanciação dos Repositórios:**
   - PessoaFisicaRepo e PessoaJuridicaRepo são instanciados para manipular dados de PessoaFisica e PessoaJuridica, respectivamente.
2. **Adicionar Dados:**
   - Criamos instâncias de PessoaFisica e PessoaJuridica, e as adicionamos aos respectivos repositórios.
3. **Persistir Dados:**
   - Utilizamos o método persistir para salvar os dados em arquivos binários.
4. **Recuperar Dados:**
   - Instanciamos novos repositórios para recuperar os dados dos arquivos e usamos o método recuperar.
5. **Exibir Dados:**
   - Chamamos o método exibir para mostrar os dados recuperados na saída padrão.

Análise e Conclusão:

Vantagens e desvantagens do uso de herança.

Vantagens:

Reutilização de Código: Permite a reutilização de membros e métodos de uma classe pai nas classes filhas.

Polimorfismo: Facilita a criação de código mais genérico e flexível, onde objetos de classes filhas podem ser tratados como objetos da classe pai.

Desvantagens:

Acoplamento: Pode levar a um acoplamento excessivo entre classes, tornando o código mais difícil de entender e manter.

Herança Múltipla Problema: Em linguagens que suportam herança múltipla, pode surgir o problema do diamante, onde uma classe é herdada por duas classes, que por sua vez são herdadas por uma terceira classe.

Uso da interface Serializable para efetuar persistência em arquivos binários.

A interface Serializable é necessária para indicar que a classe pode ser serializada, ou seja, seus objetos podem ser convertidos em uma sequência de bytes. Isso é crucial ao persistir objetos em arquivos binários. A serialização permite que os objetos sejam armazenados em um formato que pode ser posteriormente desserializado, reconstruindo os objetos originais. Sem a implementação da interface Serializable, o Java lançará uma exceção NotSerializableException durante a tentativa de serialização.

Como usar o paradigma funcional API stream no Java.

A API Stream do Java permite a utilização de programação funcional através do uso de expressões lambda, operações de ordem superior e pipeline de operações. As operações de Stream permitem realizar operações em conjuntos de dados de maneira mais concisa e expressiva.

Algumas características funcionais incluem:

✓ Operações Map e Filter: Transformação e filtragem de elementos do Stream.

✓ Operações Reduce: Redução dos elementos do Stream a um resultado único.

✓ Expressões Lambda: Uso de funções anônimas para definir operações a serem

realizadas nos elementos do Stream.

Padrão de desenvolvimento para persistência de dados em arquivos em Java.

O padrão mais comum adotado na persistência de dados em arquivos em Java é o padrão de projeto DAO (Data Access Object). O DAO é responsável por fornecer uma interface para alguns tipos de fonte de dados, como um banco de dados ou, no caso, arquivos. Isso isola o código de acesso a dados da lógica de negócios, proporcionando maior modularidade e facilitando a manutenção. O DAO define métodos para inserir, atualizar, excluir, e recuperar dados, permitindo uma abstração eficaz da camada de persistência.