**Sistema de Arquivos**

O sistema de arquivos é o sistema utilizado pelo sistema operacional para armazenar, organizar e acessar os dados que estão armazenados no computador.

**Categorias das operações de E/S em arquivos**

As operações de E/S em arquivos são divididas em duas categorias:

***Operações para acesso ao***

***sistema de arquivos***

• Obter a relação de arquivos e subdiretórios de um diretório.

• Ler propriedades de um arquivo ou diretório.

• Criar diretórios, apagar diretórios, renomear arquivos etc.

***Operações de leitura e edição de conteúdo de arquivos***

• Ler o conteúdo de arquivos.

• Alterar (gravar) dados em arquivos.

**Arquivo Texto X Arquivo Binário**

***Arquivo texto***

• Os bits representam caracteres.

• Podem ser lidos por editores de texto.

• São “legíveis” para os humanos.

***Arquivos binários***

• Os bits representam dados.

• Utilizam qualquer sequência de bytes.

• Mais eficiente de processar.

**Stream**

Stream é um objeto que tanto obtém dados de uma fonte (como teclado, arquivo, rede) como grava dados (tela, arquivo etc.).

**Persistência de dados primitivos**

**DataOutputStream**

• DataOutputStream oferece uma

série de métodos para gravação de

dados primitivos e string.

**Leitura de Arquivos**

• DataInputStream oferece uma série de métodos para leitura de dados primitivos e string.

• Após a leitura, o ponteiro é posicionado após a quantidade de bytes lidos.

**Observações**

• DataInputStream e DataOutputStream respectivamente, lê e grava tipos primitivos Java e Strings de forma independente de máquina.

• Os dados devem ser lidos na mesma ordem em que foram gravados.

• A exceção EOFException é lançada quando há a tentativa de ler dados além do fim do arquivo.

**Buffer de dados na leitura/gravação**

• A classe ***BufferedInputStream*** é utilizada para reduzir a quantidade de acessos ao dispositivo de armazenamento, tornando a leitura mais otimizada

**Editar dados de um arquivo**

• A classe ***DataOutputStream*** somente funciona com duas modalidades:

– Recriar um arquivo.

– Acrescentar dados ao final do arquivo.

• Não é possível alterar dados já gravados com ***DataOutputStream***

– Não possui skip().

**RandomAccessFile**

• O parâmetro ***mode*** pode ser:

♣ ***r*** – para acesso somente leitura.

♣ ***rw*** – para leitura e gravação.

♣ ***rws*** – para leitura e gravação. Cada gravação é aplicada imediatamente no arquivo. Causa a alteração da data de modificação do arquivo.

♣ ***rwd*** – Similar ao rws, porém somente causa a alteração da data de modificação do arquivo quando o arquivo é fechado.

• Os modos ***rws*** e ***rwd*** são mais lentos.

**Serialização**

A ***serialização*** de objetos consiste em converter os dados de estado de um objeto numa representação binária ou textual, a fim de armazená-lo num meio de armazenamento secundário.

**Observações**

• Podem ser gravados vários objetos no ***stream***.

• Para um objeto ser serializado, este deve implementar a interface ***Serializable*** – ***Serializable*** é uma interface de marcação.

• Valores de variáveis estáticas não são serializados.

• Para não serializar determinada variável de instância, introduzir o modificador ***transient***, como abaixo:

– private transient String atributo.

**Exceções lançadas**

• ***ClassNotFoundException***: Ocorre quando se tenta desserializar objeto de uma classe desconhecida.

• ***NotSerializableException***: Ocorre quando se quer serializar um objeto de uma classe que não implementa a interface ***Serializable***.

**Serialização - Versionamento**

• É possível “versionar” uma classe ao persistir um objeto.

• O versionamento garante que objetos somente possam ser reconstituídos quando recuperados a partir da mesma versão de classe que persistiu o objeto.

• Para estabelecer um número de versão, acrescentar:

– private static final long serialVersionUID = 1L;

• ***serialVersionUID*** deveria ser um identificador único para uma versão. • Quando não há declaração de ***serialVersionUID***, Java gera um número de versão.

• É recomendável que todas as classes serializáveis declarem explicitamente um valor para ***serialVersionUID***.

• A exceção ***java.io.InvalidClassException*** é lançada quando ocorre tentativa de desserializar um objeto que foi serializado com versão diferente.

**Serialização**

• Para obter a versão de uma classe, utilizar o utilitário “***serialver***” na linha de comando, fornecendo o nome da classe (incluindo seu pacote).

**StreamCorruptedException**

• Uma exceção da classe ***StreamCorruptedException*** pode ser lançada ao ler um arquivo com objetos persistidos.

• Ocorre ao ler um arquivo que foi gravado com ***FileOutputStream*** com objetivo de acrescentar dados.

• Falha ocorre porque sempre que ***ObjectOutputStream*** é criado, grava um novo cabeçalho no arquivo, constituído de 4 bytes.

**Persistência de texto**

• A classe PrintWriter é utilizada para gravar texto.

**Gravação de arquivo texto**

• Para gravação de arquivo de texto livre podemos utilizar a classe ***PrintWriter***.

• Para adicionar texto, utilizar os métodos: ***print()*** e ***println()***.

**Biblioteca de Estrutura de Dados**

• Uma “Estruturas de dados” é uma forma de armazenar e organizar dados na memória do computador.

• “Escolher a melhor estrutura de dados e algoritmos para uma tarefa particular é uma das chaves para o desenvolvimento de software de alta performance” (LIANG, 2019).

• Java possui uma biblioteca para manipular estruturas de dados, denominado de Java Collection Framework (JCF) – Embora seja chamado de “framework”, a JCF é uma biblioteca.

• As classes de manipulação de estrutura de dados estão agrupadas em dois grandes grupos: Coleções e Mapas.

**Collection**

• ArrayList implementa uma lista estática.

• LinkedList implementa uma lista dinâmica.

**O método iterator()**

• A interface Collection prevê que todas as classes concretas que a implemente, retorne um iterador, através do método ***iterator()***. Um iterador é um objeto que possibilita percorrer uma estrutura de dados.

**Ordenação de objetos de uma coleção**

• Numa lista contendo objetos da classe Aluno, quais atributos devem ser usados como critério para ordenação?

• Existem dois conceitos de ordenação em Java:

♣ Ordenação natural – é a ordenação típica de objetos de uma determinadaclasse. Cada classe possui apenas uma ordenação natural.

♣ Ordenação artificial – uma forma alternative de ordenar objetos de uma classe. Cada classe pode possuir várias ordenações artificais.

**Definindo a ordenação natural**

A ordenação natural de objetos de uma classe é especificada ao implementar a interface ***Comparable***:

O método ***compareTo()*** deve retornar:

-1: se this < outro.

0: se this for igual à outro.

+1: se this > outro.

• Para ordenar coleções, utilizar o método sort() da classe Collections.

**Definindo a ordenação artificial**

• A ordenação artificial de objetos de uma classe é especificada ao implementar a interface ***Comparator***: • O método ***compare()*** deve retornar:

-1: se o1 < o2.

0: se o1 for igual à o2.

+1: se o1 > o2.

**O que é um mapa de dispersão?**

• Um mapa armazena uma coleção de pares chave/valor.

• As chaves funcionam como índices. • Cada chave mapeia um valor.

• Permite executar busca, exclusão e atualização de dados (a partir da chave) de forma rápida.

• Também conhecido como tabela hash (haspmap), mapa de espalhamento etc.

• Usar o método ***put()*** para incluir um objeto no mapa.

• Requer informar a chave do objeto.

• A localização deve ser feita pela chave do objeto.

• Utilizar o método ***get()*** para localizar o objeto.

• Se a chave não tiver sido inserida previamente, o método ***get()*** retornará null.

**O método hashcode()**

• É responsabilidade da classe da chave implementar o método hashcode().

• O hashcode é um valor que pertence ao objeto. Trata-se de um número de 32 bits, que permite que o objeto seja utilizado numa estrutura de dados mapa de dispersão.

♣ A partir deste número é possível calcular qual a posição em que se encontra o par.

• Dois objetos iguais devem retornar o mesmo hashcode.

♣ Se dois objetos (a e b) são iguais, isto é, o a.equals(b) == true, então o método hashcode() deve retornar o mesmo valor para os dois objetos.

♣ Sempre que o método equals() é implementado, deve-se sobrescrever hashcode() também, e vice versa.

• Dois objetos distintos podem retornar o mesmo hashcode, mas se possível, deve-se evitar.