Programação para Web I
AULA 4

Profa. Silvia Bertagnolli

Pacote NIO2

Pacote nio2

API traz novas funcionalidades:

- Navegação em árvores de diretórios

- Manipulação de atributos de arquivos e diretórios

- Novos recursos para simplificar operações como mover, copiar e apagar arquivos

Novos Pacotes

java.nio.file - pacote principal da API; com classes para manipulação de arquivos

java.nio.file.attribute - disponibiliza classes e interfaces para manipular os atributos dos arquivos, como, por exemplo, data de criação, última modificação...

java.nio.file.spi - disponibiliza classes e interfaces que podem ser implementadas para dar suporte unificado a um novo sistema de arquivos

Classes

Files: manipula arquivos e diretórios de acordo com o sistema de arquivos do S.O.

FileStore: Representa os recursos de armazenamento de arquivos, como disco rígido, sistema de arquivos, partição, etc.

FileSystem: Fornece uma interface para o sistema de arquivos utilizado. Com esta interface é possível criar objetos **Path** que representam arquivos e diretórios

Classes

FileSystems: Classe utilizada para obter um sistema de arquivos. Através de seu método estático **getDefault()** é possível ter acesso ao sistema de arquivos padrão

Path: Representa um caminho para um diretório ou arquivo, substituindo a classe **File** do pacote **java.io**

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça os exercícios 10 a 12

Arquivos

Formas de Manipular Arquivos

Manipulação de entrada e saída de caracteres – transferência de textos

Buffers - melhoram a eficiência da velocidade de leitura e escrita

Manipulação de entrada e saída de bytes – transferência de dados binários

FileWriter e FileReader

Arquivos de Caracteres

Classes **FileWriter** e **FileReader** são usadas para gravar e ler um fluxo de caracteres de um arquivo

Métodos principais:

- read() – definido na classe Reader para leitura de dados de um arquivo

- write() – definido na classe Writer para escrita em arquivo

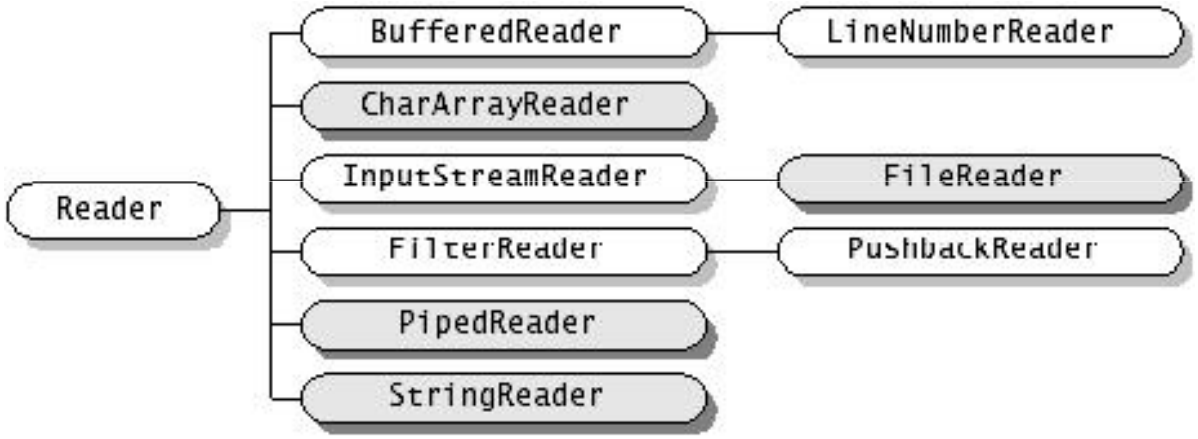
A classe *FileReader* nos fornece o método *read* que lê um único caractere do arquivo e retorna o número inteiro de seu código na tabela unicode, caso seja o fim do arquivo retornará -1

Hierarquia de Classes

Escrevendo caracteres e strings

```
File arqE= new File(“c://TesteEx1.txt");
try{
   FileWriter fw = new FileWriter(arqE) ;
   fw.write('2');
   fw.write("2");
   fw.flush();
   fw.close();
}catch(IOException e){
   System.out.println("Exceção na escrita!");
}
```

Hierarquia de Classes



Lendo caracteres e strings

```
File arqLeit = new File("TesteEx1.txt");
try{
   FileReader fr = new FileReader(arqLeit);
   int  c =   fr.read();
   while( c != -1){
        System.out.print( (char) c );
        c =  fr.read();
   }
}catch(FileNotFoundException e){
       System.out.println("Arquivo não encontrado!");
}catch(IOException e){
        System.out.println("Exceção na leitura!");
}
```

Exercícios: FileReader e FileWriter

1. Faça a leitura de palavras que devem ser gravadas em arquivo qualquer. Use para a leitura de palavras a classe JOptionPane

2. No projeto Arquivos crie a classe ManipulaArquivos que deve declarar os métodos abaixo:

    1. Método de classe gravarArquivo(String nomeArq), que lê dados com JOptionPane e grava no arquivo

    2. Método de classe lerArquivo(String nomeArq), que lê os dados do arquivo e mostra os dados lidos em janela

    3. Método main() deve chamar os métodos gravarArquivo() e lerArquivo()

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça os exercícios 1 a 3

BufferedWriter e BufferedReader

BufferedWriter e BufferedReader

As classes BufferedWriter e BufferedReader são usadas, respectivamente, para escrever e ler caracteres usando um buffer

BufferedWriter

```
File arquivo = new File("arquivo2.txt");
FileWriter fw = new FileWriter( arquivo ) ;
BufferedWriter escrita = new BufferedWriter(fw);
escrita.write( "teste" );
escrita.newLine();
escrita.write( "teste2");
escrita.flush();
escrita.close();
```

BufferedReader

```
File arquivo = new File("arquivo2.txt");
FileReader fr = new FileReader(arquivo);
BufferedReader leitura = new BufferedReader(fr);
String content;
while( ( content = leitura.readLine() ) != null){
    System.out.println( content );
}
leitura.close();
```

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça os exercícios 6 e 7

InputStream e OutputStream

InputStream e OutputStream

A classe FileOutputStream é usada para gravar bytes em um arquivo

A classe FileInputStream é usada para ler bytes de um arquivo

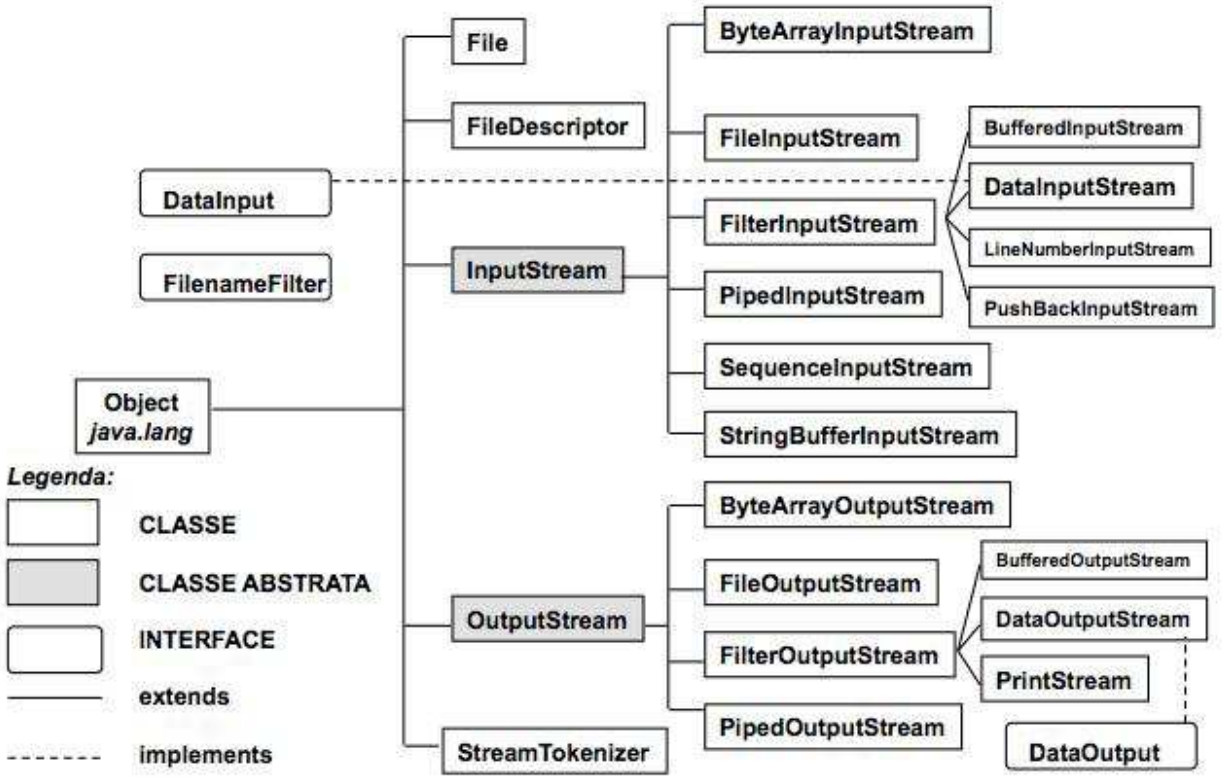
InputStream e OutputStream

Para escrever dados em um arquivo é usado o método **write** que pode receber um byte ou um vetor de bytes

O método **getBytes** converte os caracteres da String em bytes, pois a classe OutputStream precisa desse formato para que os bytes sejam gravados

O o método **read** retorna -1 se chegar no final do arquivo

Classes para Manipular Arquivos



InputStream e OutputStream

Classes para leitura e escrita de um **byte** ou de uma sequência de **bytes**

Métodos principais:

`read()` - definido na classe InputStream para leitura

`write()` - definido na classe OutputStream para escrita

Hierarquia de Classes

FileOutputStream

```
File arquivo = new File("teste.bin");
try{
        OutputStream saida=new FileOutputStream(arquivo);
        byte[] b = {50,51,52,53};
        String string = "Teste com várias palavras";
        saida.write( 53 ); saida.write( b );
        saida.write( string.getBytes() );
        saida.flush(); saida.close();
}catch(SecurityException e){System.out.println("Exc segurança!");
}catch(FileNotFoundException e){
            System.out.println("Arq não encontrado!");
}catch(IOException e){System.out.println("Exceção na escrita!");
}
```

FileInputStream

```
try{
  InputStream entrada = new FileInputStream(arquivo);
  int content=0;
  while ( (content = entrada.read() ) != -1) {
    System.out.println( content +" - "+ ( (char) content) );
  }
  entrada.close();
}catch(SecurityException e){ System.out.println("Exc seg");
}catch(FileNotFoundException e){
      System.out.println("Arquivo não encontrado!");
}catch(IOException e){System.out.println("Exceção escrita!");
}
```

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça os exercícios 4, 5 e 8

TRY COM RECURSOS

Sem try com recursos

```
InputStream in = null;
OutputStream out = null;
     try {
       in = new FileInputStream(origem);
       out = new FileOutputStream(destino);
       byte[] buf = new byte[8192]; int n;
       while ((n = in.read(buf)) >= 0) out.write(buf, 0, n);
    } catch (FileNotFoundException | IOException ex) {
       System.out.println("Problemas com a cópia: " + ex);
    } finally {
```

→

Sem try com recursos

```
} finally {
          if (in != null)
              try {
                  in.close();
              } catch (IOException ex) {
                 //tratamento
              } finally {
                 if (out != null)
                     try {
                         out.close();
                     } catch (IOException ex) {
                        // tratamento ...
                     }
              }
```

TRY COM RECURSOS

```
try (InputStream in = new FileInputStream(origem);
     OutputStream out = new FileOutputStream(destino);)
{
          byte[] buf = new byte[8192];
          int n;
          while ((n = in.read(buf)) >= 0)
             out.write(buf, 0, n);
} catch (FileNotFoundException ex) {
    System.out.println("Problemas com a cópia: " + ex);
}
```

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle refaça todos os exercícios que abordam o tema de arquivos usando try com recursos

ObjectOutputStream e ObjectInputStream

ObjectOutputStream e ObjectInputStream

A classe ObjectInputStream permite ler objetos de um arquivo no formato binário

A classe ObjectOutputStream permite gravar objetos em um arquivo no formato binário

Classes são usados para a leitura e a gravação de objetos serializados

Gravando Objetos

```
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream( nomeArq) );

Pessoa objPessoa = new Pessoa();

out.writeObject(objPessoa);

out.flush();

out.close();
```

Lendo Objetos

```
ObjectInputStream in = new ObjectInputStream(new
FileInputStream( nomeArq) );

objPessoa = (Pessoa) in.readObject();

System.out.println(objPessoa);

in.close();
```

Serializando Objetos

```
public class Pessoa implements Serializable {

      //....

}
```

Para que um objeto possa ser armazenado em um arquivo no formato correto é necessário que a sua classe implemente a interface Serializable

EXERCÍCIOS

Exercícios

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça a Questão 13.

Manipulando Data e Hora

Nova API Data e Hora (Java 8)

API que fornece manipulação mais simples para trabalhar com objetos que armazenam: só data, só horas ou ambas

Pacote java.time

Classes:

- LocalDate
- LocalTime
- Period
- Instant
- ZoneDateTime
- Outras

Classe	Descrição
Clock	É usada para acessar o instante atual de tempo, data e hora por meio de uma zona de tempo
Duration	É usada para medir a quantidade de tempo a cada 34,5 segundos
Instant	É usada para definir um marco de tempo a ser medido durante a execução do código
LocalDate	É usada para definir a data do sistema no formato ISO (AAAA-MM-DD)
LocalDateTime	É usada para definir a data e hora do sistema no formato ISO (AAAA-MM-DDTHH:MM:SS) sem uso do fuso horário
LocalTime	É usada para definir um tempo em horas, minutos e segundos do sistema no formato ISO (HH:MM:SS) sem uso do fuso horário

Classe	Descrição
MonthDay	É usada para o mês e o dia de calendário no formato ISO (--MM-DD)
OffsetDateTime	É usada para definir a data e hora do sistema com deslocamento de fuso horário UTC/Greenwich no formato (AAAA-MM-DDTHH:MM:SS+HH:MM)
OffsetTime	É usada para definir a hora do sistema com deslocamento de fuso horário UTC/Greenwich no formato (HH:MM:SS+HH:MM)
Period	É usada para medir a quantidade de tempo baseada no calendário do sistema
Year	É usada para definir um ano do calendário no sistema no formato (AAAA)
YearMonth	É usada para definir um ano e mês do calendário no sistema no formato (AAAA-MM)

Classe	Descrição
ZonedDateTime	É usada para definir a data e hora do sistema com deslocamento de fuso horário no formato ISO (AAAA-MM-DDTHH:MM:SS+HH:MM) para certa região do globo terrestre
ZoneId	É usada para definir um ID para a identificação do fuso horário para certa região do globo terrestre
ZoneOffser	É usada para definir um deslocamento de fuso horário padrão UTC/Greenwich como +MM:MM
DayOfWeek (enumeração)	É usada para definir um dia da semana como “Segunda-feira”
Month (enumeração)	É usada para definir um mês do calendário como “Agosto”
DateTimeException	É usada para interceptar um problema que ocorra com os cálculos de datas e horas

LocalDate

LocalDate

Usada para manipular datas - representa uma data em um período de 24 horas com dia, mês e ano definidos

Com essa classe é possível:

- Definir uma data específica

- Calcular o número de dias entre duas datas

- Obter informações sobre a data, ver próxima data

LocalDate

Método	Descrição
getDayOfWeek()	Usada para retornar o nome do dia da semana da data relacionada
getDayOfMonth()	Usada para retornar o dia do mês da data relacionada
getMonth()	Usada para retornar o nome do mês da data relacionada
getMonthValue()	Usada para indicar o número do mês da data relacionada
getYear()	Usada para apresentar o número do ano da data relacionada
getDayOfYear()	Usada para indicar o número de dias percorridos junto à data indicada a partir de 1 de janeiro da data relacionada

Exemplo

```
public class ClasseLocalDate {
   public static void main(String[] args) {
        LocalDate dataAtual = LocalDate.now();
        System.out.println("Data: " + dataAtual);

        //criar um LocalDate para uma data específica
        //data = 10/03/2017
        LocalDate dataEspecifica = LocalDate.of(2017, 3, 10);
        System.out.println("Data: " + dataEspecifica);

        System.out.println("Data - dia do mês: " +
                                  dataEspecifica.getDayOfMonth());

  }
}
```

LocalTime

LocalTime

A classe LocalTime serve para representar apenas um horário, sem data específica

Com essa classe é possível:

- Definir um horário específico
- Calcular se um horário está antes ou depois de outro
- Somar e subtrair horas, minutos e segundos em um horário

Exemplo

```
public class ClasseLocalTime {
    public static void main(String[] args) {
       //horário atual
       LocalTime hAtual = LocalTime.now();
       System.out.println("Hora: " + hAtual );

       //definindo hora específica
       LocalTime horarioDeEntrada = LocalTime.of(19, 0);
       System.out.println(horarioDeEntrada); //19:00

       //horário atual
       System.out.println("Hora: minutos: " + hAtual.getMinute());
       //segundos do dia com relação à data
       System.out.println("Segundos do dia: "+hAtual.toSecondOfDay());
    }
}
```

LocalDateTime

LocalDateTime

A classe LocalDateTime é usada para representar uma data e um horário específicos

Com essa classe é possível:

- Calcular um período entre datas
- Permite obter dados da data e da hora

Exemplo

```
public class ClasseLocalDateTime {
   public static void main(String[] args) {
     LocalDateTime dataHoraAtual = LocalDateTime.now();
     System.out.println("Data e Hora: " + dataHoraAtual);
     System.out.println("Data - dia do ano: " +
                                        dataHoraAtual.getDayOfYear());
   }
}
```

Period x Duration

Period x Duration

Period
- Classe usada para calcular o tempo usando anos, meses e dias
Possibilita calcular a diferença entre datas usando o método between()
- Fornece métodos como: getDays(), para recuperar a quantidade de dias; getMonths(), para a quantidade de meses; e getYears(), para obter a quantidade de anos

Duration
- Classe usada para calcular o tempo usando horas, minutos, segundos, nanosegundos

Exemplo

```
public class ClassePeriodoDuracao {
  public static void main(String[] args) {
    //usando Period
    LocalDate homemNoEspaco = LocalDate.of(1961, Month.APRIL, 12);
    LocalDate homemNaLua = LocalDate.of(1969, Month.MAY, 25);
    Period periodo = Period.between(homemNoEspaco, homemNaLua);
    System.out.printf("%s anos, %s mês e %s dias =",
        periodo.getYears(), periodo.getMonths(), periodo.getDays());
    //usando Duration
    LocalDateTime inicio = LocalDateTime.of(2016, 10, 20, 0, 0, 0);
    LocalDateTime fim = LocalDateTime.of(2017, 2, 16, 0, 0, 0);
    Duration dur = Duration.between(inicio, fim);
    System.out.printf(“%s dias %s horas, %s minutos e %s segundos",
       dur.toDays(), dur.toHours(), dur.toMinutes(), dur.getSeconds());
  }
}
```

Formatando Datas

DateTimeFormatter

Essa classe possui métodos para formatar a data e o horário

Ela também possui métodos que permitem formatar somente datas válidas

Acesse o link abaixo para ver todos os padrões que podem ser usados para formatar uma data e um horário:

https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

DateTimeFormatter

```
public class ClasseDateTimeFormatter {
  public static void main(String[] args) {
    LocalDateTime dataHora = LocalDateTime.now();
    System.out.println(dataHora);
    System.out.println(dataHora.format(
                  DateTimeFormatter.ofPattern("dd/MM/yyyy hh:mm:ss")));

    //forçar que uma data seja validada
    DateTimeFormatter formato =
                            DateTimeFormatter.ofPattern("dd/MM/yyyy");
    formato = formato.withResolverStyle(ResolverStyle.STRICT);
    LocalDate data = LocalDate.of(2017, 2, 31);
    System.out.println(dataHora.format(formato));
  }
}
```

Trabalho

Trabalho

Usando o arquivo ListaExercicios_Aula4 que está disponível no Moodle faça as Questões 22 e 23.