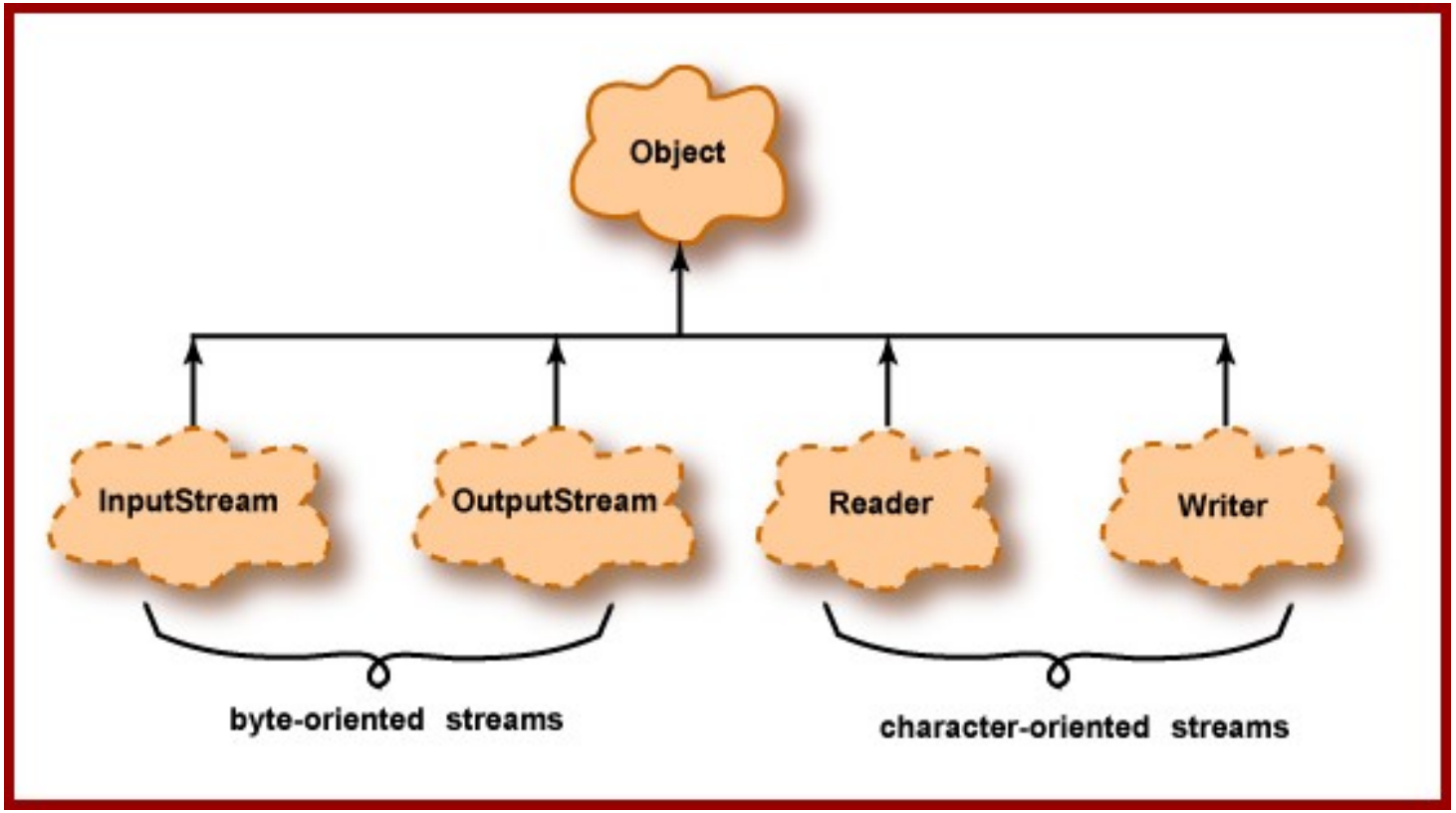
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Leitura e Escrita de Texto em Arquivos usando Java

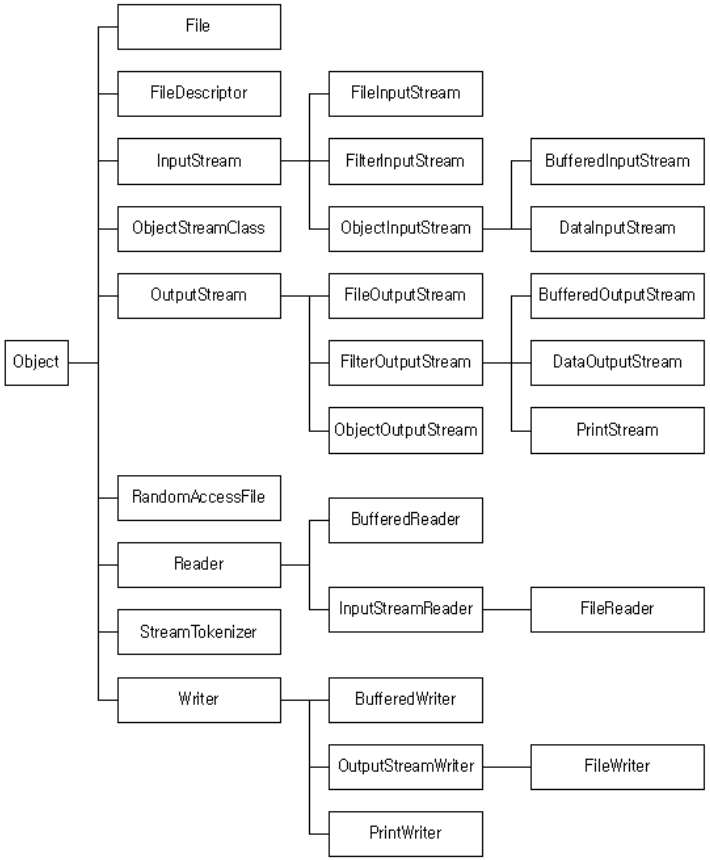
Programação Orientada a Objetos

Prof. Fabrício M. Lopes
fabricio@utfpr.edu.br

Classes Java para Entrada e Saída



Classes Java para Entrada e Saída



Classes Java para Entrada e Saída

- As classes **Reader** e **Writer** foram projetadas para processar a entrada e saída de texto.

- As classes adequadas para a escrita e a leitura em arquivos texto são: **FileWriter** e **FileReader**.

java.io.FileReader

- Classe para a leitura de arquivos texto.
- Construtores:
  - FileReader(***File*** file)
    - Cria um objeto FileReader, dado o arquivo a ser lido.
  - FileReader(***String*** fileName)
    - Cria um objeto FileReader, dado o nome de um caminho para um arquivo a ser lido, deve ser um caminho completo.

java.io.FileWriter - Métodos construtores

- FileWriter(***File*** file)
  - Constrói um objeto FileWriter dado um objeto File.
- FileWriter(***File*** file, ***boolean*** append)
  - Constrói um objeto FileWriter dado um objeto File com um booleano indicando se deve continuar a escrita (true) ou criar um novo arquivo (false).
- FileWriter(***String*** fileName)
  - Constrói um objeto FileWriter dado um caminho para um arquivo.
- FileWriter(***String*** fileName, ***boolean*** append)
  - Constrói um objeto FileWriter dado um caminho para um arquivo e um booleano indicando se deve continuar a escrita (true) ou criar um novo arquivo (false).

Exemplo: LeituraTexto01.java

```
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;
/**
 * @author fabricio@utfpr.edu.br
 */
public class LeituraTexto01 {
    public static void main(String[] args) {
        Reader fileReader = null;
        try {
            fileReader = new FileReader("/home/fabricio/wang2007.csv");
            int data = fileReader.read();
            while (data != -1) {
                char c = (char) data;
                //do something with data...
                System.out.println(data +" = "+ c);
                data = fileReader.read();
            }
            fileReader.close();
        } catch (IOException erro){
            erro.printStackTrace();
        }
    }
}
```

Exemplo: EscritaTexto01.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
/**
 * @author fabricio@utfpr.edu.br
 */
public class EscritaTexto01 {
    public static void main(String[] args) {
        Writer fileWriter = null;
        try {
            fileWriter = new FileWriter("/home/fabricio/log.txt", false);
            fileWriter.write("Hello world!");
            fileWriter.close();
        } catch (IOException erro) {
            erro.printStackTrace();
        }
    }
}
```

java.io.BufferedReader

- Lê texto a partir de um fluxo de entrada de caracteres, bufferizando os caracteres de modo a proporcionar a leitura eficiente de caracteres, matrizes e linhas.
- Em geral, cada solicitação de leitura feita por um Reader faz com que seja feita uma leitura correspondente do caractere. Logo, é importante usar um buffer para tornar a leitura mais eficiente. Exemplo:
    - BufferedReader entrada = new BufferedReader(new FileReader("log.txt"));

java.io.BufferedWriter

- Escreve texto para um fluxo de saída de caracteres, bufferizando os caracteres de modo a proporcionar a escrita eficiente de caracteres e strings.
- O método newLine(), que utiliza o separador de linha definido pelo sistema operacional line.separator. Dado que nem todas as plataformas usam o caractere ('\n') para terminar linhas.
- Em geral, um Writer faz a escrita imediatamente do caractere. Logo, é importante usar um buffer para tornar a escrita mais eficiente. Exemplo:
  - ***BufferedWriter saida = new BufferedWriter(new FileWriter("log.txt",false)));***

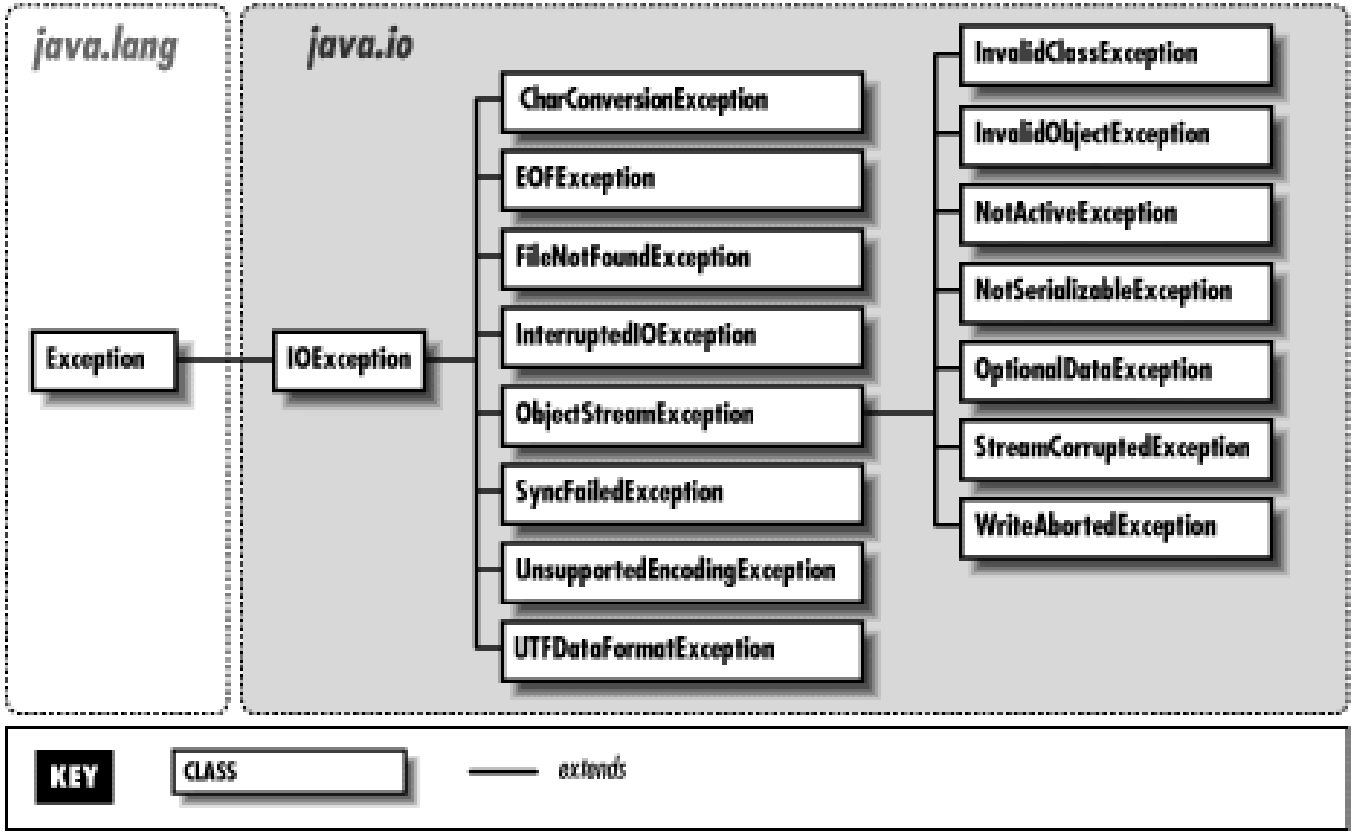
Exemplo: LeituraTexto02.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
/**
 * @author fabricio@utfpr.edu.br
 */
public class LeituraTexto02 {
    public static void main(String[] args) {
        BufferedReader fileReader = null;
        try {
            fileReader = new BufferedReader(new FileReader("/home/fabricio/log.txt"));
            while (fileReader.ready()) {
                String data = fileReader.readLine();
                System.out.println(data);
            }
            fileReader.close();
        } catch (IOException erro){
            erro.printStackTrace();
        }
    }
}
```

Exemplo: EscritaTexto02.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
/**
 * @author fabricio@utfpr.edu.br
 */
public class EscritaTexto02 {
    public static void main(String[] args) {
        BufferedWriter fileWriter = null;
        try {
            fileWriter = new BufferedWriter(new FileWriter("/home/fabricio/log.txt", false));
            fileWriter.write("Hello world!");
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException erro) {
            erro.printStackTrace();
        }
    }
}
```

Tratamento de Exceções



Tratamento de Exceções

- Em linhas gerais as exeções devem ser tratadas na linguagem Java, caso contrário uma mensagem de erro é impressa, e o programa é encerrado.
- Existem basicamente dois mecanismos para isso:
  - Tratamento de exceções com ***try/catch***;
  - Delegar o tratamento de exceções com ***throws***

Tratamento de Exceções – java.io

```
try {
        //operações que podem gerar exceções …

    } catch (EOFException erro) {

        //tratamento das exceções, iniciando pela mais específica e seguindo pelas mais gerais

    } catch (IOException erro){

        //tratamento para a segunda exceção...

    } finally {
        //bloco sempre executado, ocorrendo ou não exceções...
    }
}
```

Tratamento de Exceções – Exemplo java.io

```
public void escreveTexto(String filename) throws FileNotFoundException
{
   File entrada = new File(filename);
   FileWriter escritor = new FileWriter(entrada, false);
   escritor.write("Hello world!");
   escritor.close();
}
```

A cláusula ***throws*** sinaliza ao chamador do método que ele pode encontrar um *FileNotFoundException*. Então, o chamador precisa tomar a decisão de tratar essa exceção, ou delegar novamente que a exceção pode ocorrer.

Exercício Acompanhado

1. Escreva um programa que leia um arquivo texto. Leia cada linha e escreva em um arquivo de saída, precedido por números de linha.
   - Exemplo Entrada:

     “Maria era programadora
     trabalhava em sua casa
     adorava o seu trabalho.”
   - Exemplo saída:

     “/* 1 */ Maria era programadora
     /* 2 */ trabalhava em sua casa
     /* 3 */ adorava o seu trabalho.”
   - Pergunte ao usuário quais os arquivos de entrada e de saída.

Referências

- DEITEL, P.J. Java - Como Programar. Porto Alegre: Bookman, 2001.
- NIEMEYER, Patrick. Aprendendo java 2 SDK. Rio de Janeiro: Campus, 2000.
- MORGAN, Michael. Java 2 para Programadores Profissionais. Rio de Janeiro: Ciência Moderna, 2000.
- HORSTMANN, Cay, S. e CORNELL, Gary. Core Java 2. São Paulo: Makron Books, 2001 v.1. e v.2.