PREPARAÇÃO PARA MARATONA DE PROGRAMAÇÃO
CETEC CPS 2016

1ª Parte

Nesta primeira parte do nosso treinamento, veremos como abrir arquivos de texto utilizando algumas das bibliotecas do Java.

Os códigos fontes aqui apresentados, bem como os arquivos textos de exemplos, encontram-se disponíveis na plataforma. Para um melhor entendimento, sugere-se a execução dos programas. Caso escolha utilizar alguma IDE, como o NetBeans ou Eclipse, é necessário salvar o arquivo txt na pasta principal do projeto.

Byte Streams

Programas em Java utilizam *byte streams* para implementar entradas e saídas de 8-bits (bytes). Todas as classes byte stream são descendentes das classes InputStream e OutputStream.

Para demonstrar com a byte streams funciona, vamos focar na parte de arquivos de I/O byte streams, FileInputStream e FileOutputStream.

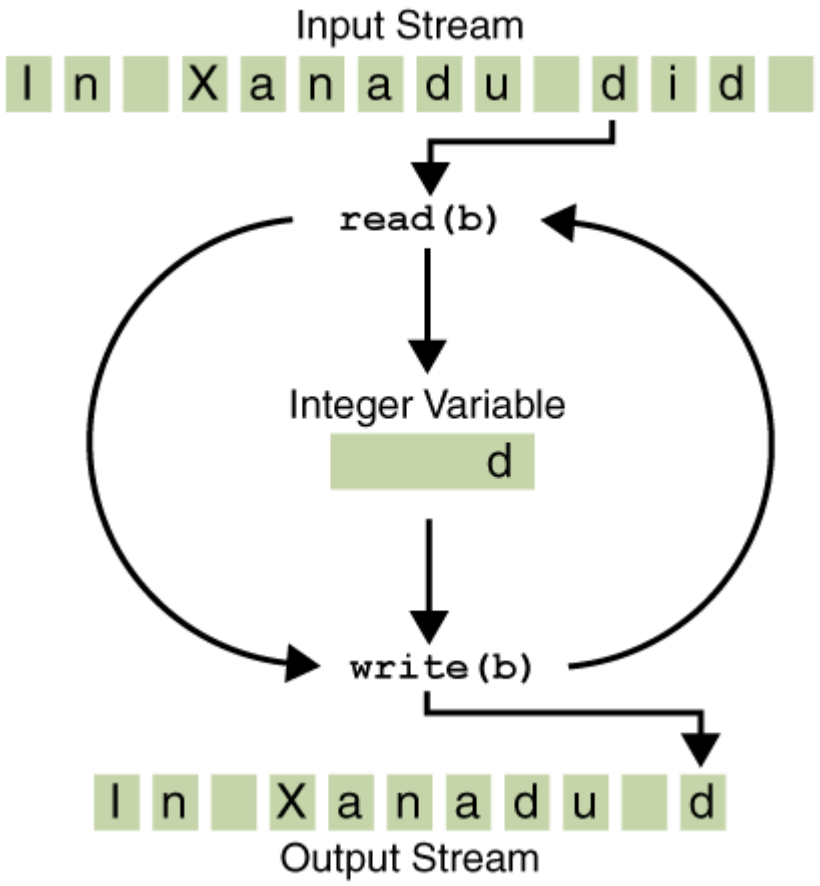
Utilizando Byte Streams

Vamos explorar FileInputStream e FileOutputStream examinando um exemplo de programa chamado CopyBytes, que utiliza byte streams para copiar o arquivo xanadu.txt, byte a byte.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }

        } finally {
```

```
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Note que o programa CopyBytes gastou a maior parte do tempo num simples loop que lê uma cadeia de caracteres de entrada e a escreve na saída, um byte de cada vez, como mostrado na figura a seguir:



Simple byte stream input and output.

Perceba que read() retorna um valor int. Assim, utilizando int como valor de retorno, permite que o read() use -1 para indicar que foi alcançado o final do arquivo.

Sempre Fechar Streams

Utilizar sempre o close() no final dos programas de Byte Streams para que tanto a Stream de entrada quanto a de saída, possam ser posteriormente acessadas pelo sistema operacional ou outros programas.

Quando não usar Byte Streams

Byte streams deverá ser utilizado apenas para os mais baixos níveis de acesso de I/O.

Character Streams

A plataforma Java armazena caracteres utilizando a convenção Unicode. Character stream I/O automaticamente traduz este formato interno para o da tabela interna local de caracteres. Em alguns países a tabela de caracteres local geralmente utiliza um conjunto de 8-bits de ASCII.

Um programa que utiliza character streams ao invés da byte streams automaticamente adapta o conjunto de caracteres locais e prontamente faz a internacionalização – tudo isso sem esforço extra do programador.

Usando Character Streams

Todas as classes Stream são decendentes das classes Reader and Writer. Assim como a byte streams, a classe character stream especializa-se em I/O de arquivos: FileReader e FileWriter. O exemplo CopyCharacters ilustra essas classes.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
             if (inputStream != null) {
```

```
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

CopyCharacters é muito similar ao CopyBytes. A mais importante diferença é que o CopyCharacters utiliza FileReader e FileWriter para entrada e saída no lugar de FileInputStream e FileOutputStream. Perceba que ambos CopyBytes e CopyCharacters utilizam uma variável int para ler e escrever de um arquivo.

Line-Oriented I/O

Caracteres I/O geralmente ocorrem em grandes conjuntos de dados ao invés de um único caractere. Uma unidade comum são as linhas: um conjunto de caracteres (String) com um caractere de final de linha ao final. Um caractere de final de linha pode ser um carriage-return/line-feed sequence ("\r\n") (retorno de carro/pula linha), um simples carriage-return ("\r") (retorno de carro), ou um simples line-feed ("\n") (pular linha ou nova linha). Suportar todas as possibilidades de terminação de linha permite aos programas ler arquivos de textos criados na maior parte dos aplicativos e sistemas operacionais.

Vamos modificar o exemplo CopyCharacters para que utilize line-oriented I/O. Para fazer isso, temos duas classes que não vimos anteriormente: BufferedReader e PrintWriter.

O exemplo CopyLines invoca BufferedReader.readLine e PrintWriter.println para fazer a entrada e saída de uma linha de texto a cada vez.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

public class CopyLines {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;

        try {
            inputStream =
                new BufferedReader(new FileReader("xanadu.txt"));
                                            outputStream =
```

```
                new PrintWriter(new
FileWriter("characteroutput.txt"));

            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

A utilização do readLine retorna uma linha de texto do arquivo aberto. O CopyLines escreve cada linha utilizando println, que acrescenta um terminador de linha para o sistema atual. Pode acabar não sendo o mesmo terminador de linha que foi utilizado no arquivo de entrada.

Há diversas maneiras de estruturar textos de entrada e saída através de caracteres e linhas. Veremos mais sobre o assunto nas próximas etapas do treinamento.

Durante a Maratona, os alunos receberão exemplos de arquivos de textos de entradas, que deverão ser processadas para a solução dos problemas apresentados, gerando arquivos de textos de saídas, com as soluções encontradas.

Em cada problema apresentado, os nomes dos arquivos de entradas e de saídas, são fornecidos.