PENGENALAN I/O

OBJEKTIF :

1. Mahasiswa Mampu Memahami Komponen *Input* dan *Output* pada Java.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA dalam Pembuatan Program Terkait *Input* dan *Output* dengan Bahasa Pemrograman Java.

PENDAHULUAN

Program komputer memiliki tiga komponen utama, yaitu: *input*, proses, dan *output*. *Input* merupakan nilai yang dimasukan ke dalam program. Proses dalam program merupakan aktivitas yang dilakukan untuk mengelola input. *Output* merupakan hasil dari pengolahan proses. Pada Java, untuk mendukung standar I/O nya, terdapat sebuah paket `java.io` yang berfungsi untuk memberikan *input* melalui *keyboard* dan kemudian menghasilkan *output* melalui layar komputer. Java menyediakan dua jenis metode I/O, yaitu metode *console* dan berbasis *byte*. Pada bab ini akan dibahas mengenai I/O dengan berbasis *byte*, yaitu `InputStream` dan `OutputStream`. `InputStream` digunakan untuk membaca atau mengambil data dari sumber. `OutputStream` digunakan untuk menulis keluaran data ke tujuan. *Byte Stream* ini merupakan metode yang digunakan untuk membaca dan menulis *raw bytes* secara satu persatu dari/ke perangkat eksternal.

13.1 FILEINPUTSTREAM

`FileInputStream` merupakan *subclass* dari *superclass* `InputStream`. `FileInputStream` digunakan untuk membaca data dari sebuah *file* tertentu yang berupa urutan *byte*. Data yang terdapat pada *file* tersebut akan dibaca secara *byte* demi *byte* dengan menggunakan method `read()`. `FileInputStream` biasa digunakan untuk membaca *raw byte* secara terurut seperti data gambar, video, audio, dan lainnya. `FileInputStream` dapat juga digunakan untuk membaca data dengan jenis karakter, namun untuk membaca data dengan jenis karakter dapat dilakukan menggunakan kelas `FileReader`. Berikut ini merupakan sintaks umum untuk deklarasi `FileInputStream`:

```
FileInputStream variabel_input = new FileInputStream(name:"file_path");
```

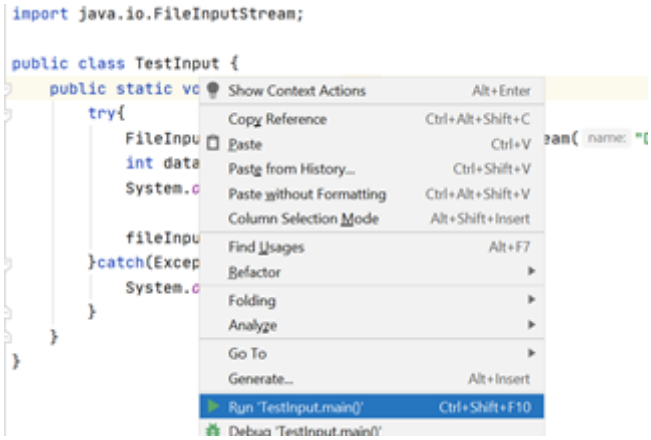
`FileInputStream` dideklarasikan seperti membuat objek baru dengan menggunakan operator new(). variabel_input merupakan nama variabel yang diberikan untuk pembuatan objek `FileInputStream` baru. `file_path` merupakan *file* yang akan dipanggil dan dibaca datanya. Berikut ini merupakan contoh program dengan implementasi `FileInputStream` membaca sebuah data gambar:

```
package com.integratedlaboratory.program;

import java.io.FileInputStream;

public class TestInput {
```

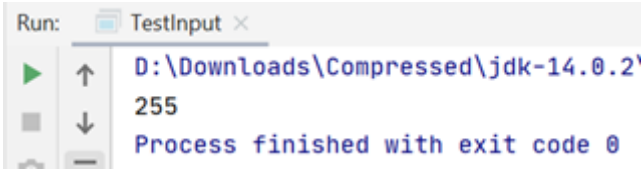
```
    public static void main(String args[]){
        try{
                                            FileInputStream    fileInput    =    new
FileInputStream("D:/Photo/logo.jpg");
            int data = fileInput.read();
            System.out.print(data);

            fileInput.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:

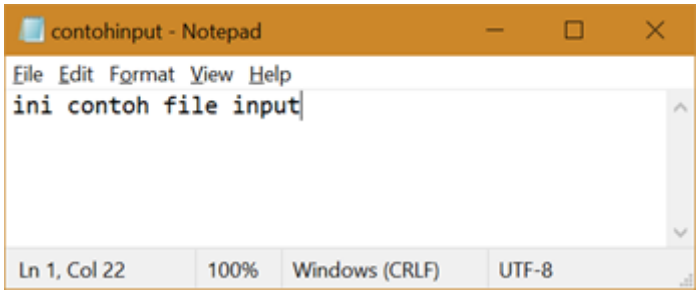


Hasil program:



Pada program di atas, sebelumnya sudah terdapat sebuah data gambar logo.jpg pada direktori D:\Photo. Data gambar tersebut dideklarasikan direktori nya di dalam objek baru `FileInputStream`. Data gambar pada program di atas, akan dibaca dalam bentuk *byte*, sehingga keluaran yang dihasilkan adalah 255 *byte*. Berikutnya merupakan contoh program dengan implementasi `FileInputStream` yang akan membaca data dengan jenis karakter:

Sebelumnya, buat sebuah *file* dengan nama contohinput.txt pada teks editor:



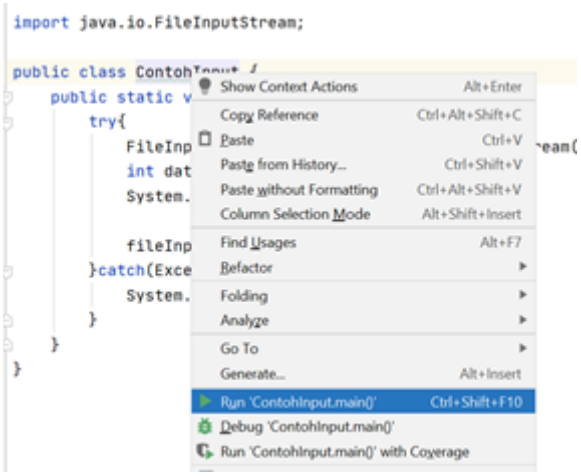
Setelah itu, implementasi pada program:

```
package com.integratedlaboratory.program;
```

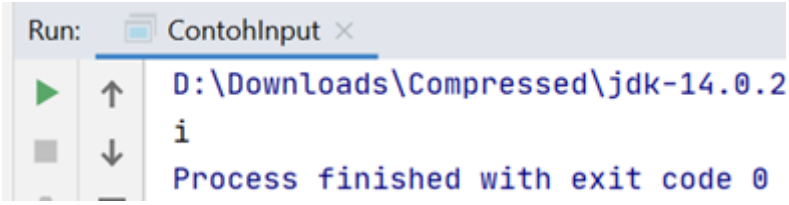
```
import java.io.FileInputStream;

public class ContohInput {
    public static void main(String args[]){
        try{
                                            FileInputStream    fileInput    =    new
FileInputStream("D:/Notes/contohinput.txt");
            int data = fileInput.read();
            System.out.print((char) data);

            fileInput.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:



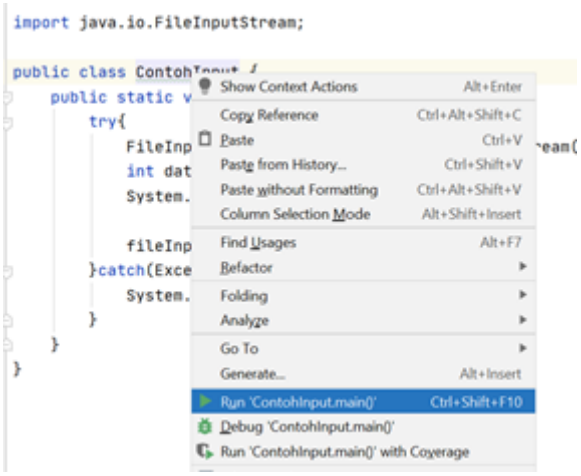
Pada contoh program di atas, `FileInputStream` dilakukan di dalam blok try dan catch. Pada catch terdapat exception dikarenakan *class* program ini tidak mengextend `IOException`. `FileInputStream` dibuat objek baru dengan nama variabelnya adalah fileInput dan file_path yang akan dibaca adalah *file* contohinput.txt yang terdapat pada direktori D:/Notes. `fileInput.read();` mendeklarasikan method read ke dalam variabel int data. `System.out.print((char) data);` akan menampilkan data yang dibaca dalam bentuk karakter. Jika char dihilangkan, maka keluaran data yang akan dibaca masih dalam bentuk *byte*. Pencetakkan data dilakukan *byte* demi *byte* atau huruf perhuruf setiap fungsi `System.out.print`. Berikut ini merupakan bukti bahwa data yang dibaca oleh `FileInputStream` adalah *byte* demi *byte*:

```
package com.integratedlaboratory.program;
```

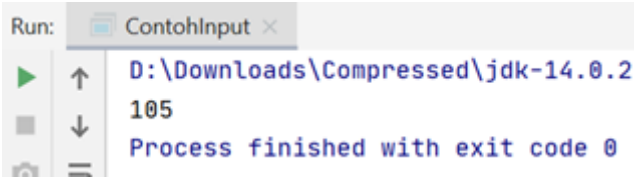
```
import java.io.FileInputStream;

public class ContohInput {
    public static void main(String args[]){
        try{
                                            FileInputStream    fileInput    =    new
FileInputStream("D:/Notes/contohinput.txt");
            int data = fileInput.read();
            System.out.print(data);

            fileInput.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:



Pada program di atas, deklarasi char dalam `System.out.print((char) data);` dihilangkan. Sehingga keluaran yang ditampilkan adalah *byte* 105 yang merupakan konversi dari huruf "i". Dari contoh program di atas, dapat diambil kesimpulan bahwa `FileInputStream` akan membaca data secara *byte* demi *byte*.

13.2 FILEOUTPUTSTREAM

`FileOutputStream` merupakan *subclass* dari `OutputStream`. `FileOutputStream` digunakan untuk menulis data dari sebuah *file* tertentu yang berupa urutan *byte*. Data yang terdapat pada file tersebut akan ditulis secara *byte* demi *byte* dengan menggunakan method `write()`. `FileOutputStream` biasa digunakan untuk membaca *raw byte* secara terurut seperti data gambar, video, audio, dan lainnya. Berikut ini merupakan sintaks umum dari `FileOutputStream`:

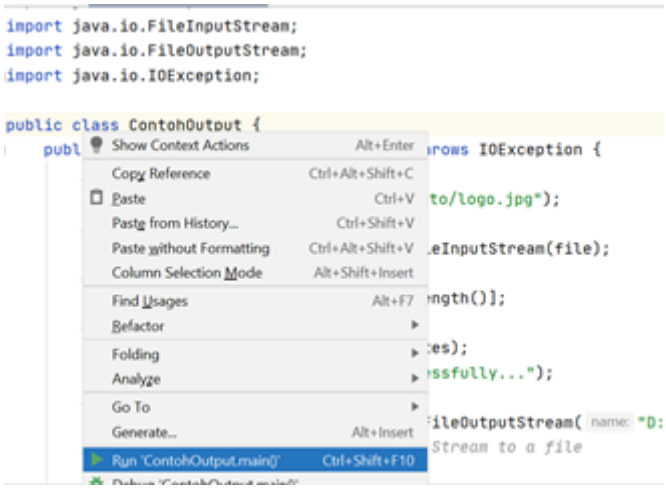
```
FileOutputStream variabel_output = new FileOutputStream(name:"file_path")
```

`FileOutputStream` dideklarasikan seperti membuat objek baru dengan menggunakan operator `new()`. variabel_output merupakan nama variabel yang diberikan untuk pembuatan objek `FileOutputStream` baru. `file_path` merupakan *file* yang akan dipanggil dan ditulis datanya. Berikut ini merupakan contoh program dengan implementasi `FileOutputStream` menulis sebuah data gambar:

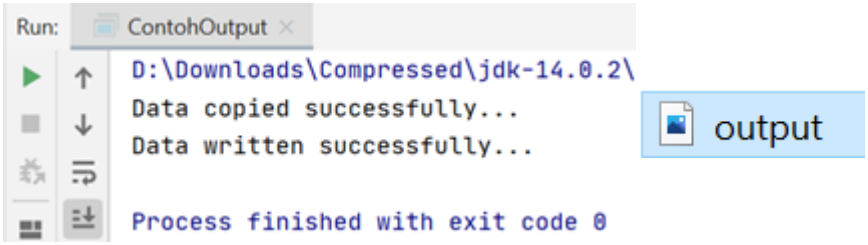
```
package com.integratedlaboratory.program;

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

public class ContohOutput {
    public static void main(String args[]) throws IOException {
        File file = new File("D:/Photo/logo.jpg");
        FileInputStream inputStream = new FileInputStream(file);
        byte bytes[] = new byte[(int) file.length()];
        int numOfBytes = inputStream.read(bytes);
        System.out.println("Data copied successfully...");

                                    FileOutputStream    outputStream    =    new
FileOutputStream("D:/Photo/output.jpg");
        outputStream.write(bytes);
        System.out.println("Data written successfully...");
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:

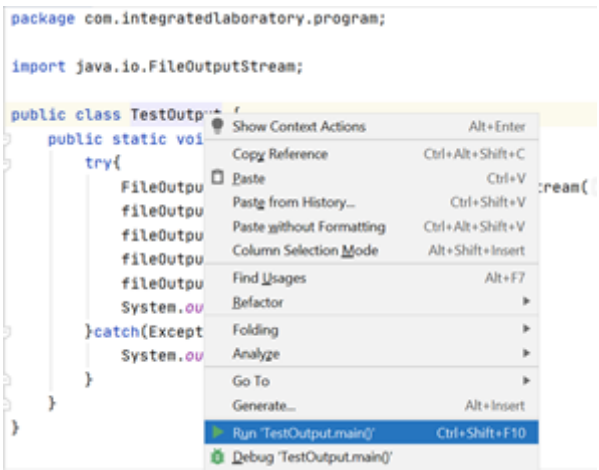


Pada contoh program di atas, dideklarasikan direktor *file* yang akan dibaca terlebih dahulu pada objek *file* baru. Lalu dideklarasikan `FileInputStream` terlebih dahulu pada objek baru *file* yang akan ditulis nantinya. Setelah *file* berhasil dibaca, baru akan dilakukan deklarasi `FileOutputStream` dengan direktori *file* yang sama, namun pada contoh program ini dibedakan nama *file* nya agar terlihat apakah *file* benar berhasil tertulis seperti pada blok `FileInputStream`. Jika data telah berhasil dibaca dan ditulis kembali, maka pada direktori yang telah diberikan, akan muncul *file* baru dengan nama yang telah disesuaikan. Berikutnya adalah contoh `FileOutputStream` untuk menulis *file* dengan jenis karakter:

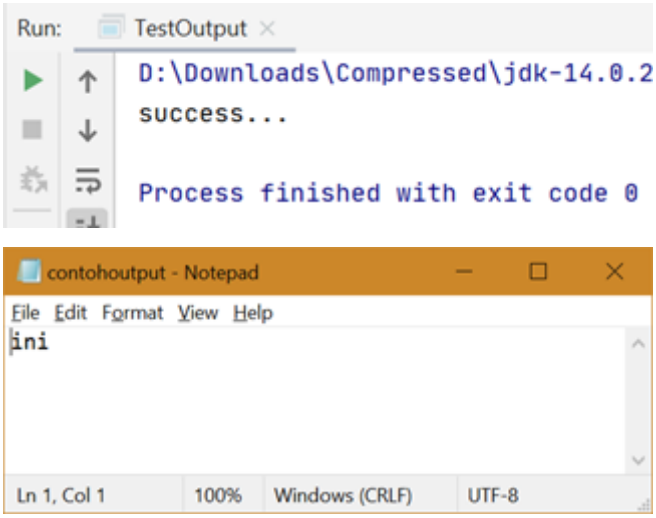
```
package com.integratedlaboratory.program;

import java.io.FileOutputStream;

public class TestOutput {
    public static void main(String args[]){
        try{
                                            FileOutputStream    fileOutput    =    new
FileOutputStream("D:/Notes/contohoutput.txt");
            fileOutput.write(105);
            fileOutput.write(110);
            fileOutput.write(105);
            fileOutput.close();
            System.out.println("success...");
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



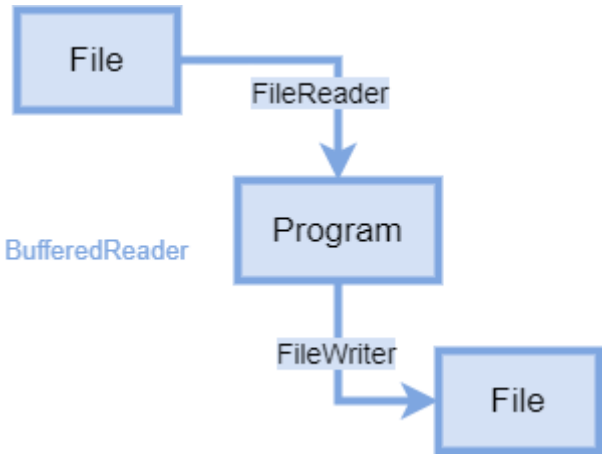
Hasil program:



Pada contoh program di atas, `FileOutputStream` dideklarasikan pada objek baru dengan direktori D:/Notes/contohoutput.txt, dimana sebelumnya *file* contohoutput.txt belum pernah dibuat. Pada contohoutput.txt dituliskan isi dalam bentuk *byte* yaitu 105, 110, dan 105 dengan method `write()`. Jika *file* telah berhasil ditulis, maka akan tampil keluaran "success...".

13.3 BUFFEREDINPUTSTREAM

`BufferedInputStream` merupakan *subclass* dari *class* `java.io` yang berfungsi untuk membaca per karakter dan mengubahnya menjadi `integer` dari sebuah `InputStream` dengan menggunakan method `FileReader`. Pada `BufferedInputStream`, karakter dapat dibaca secara baris per baris dalam satuan *byte* dengan menggunakan method `readline()` yang memiliki kemampuan membaca data lebih cepat karena data dibaca melalui memori. Berikut ini merupakan ilustrasi dari implementasi `BufferedReader`:



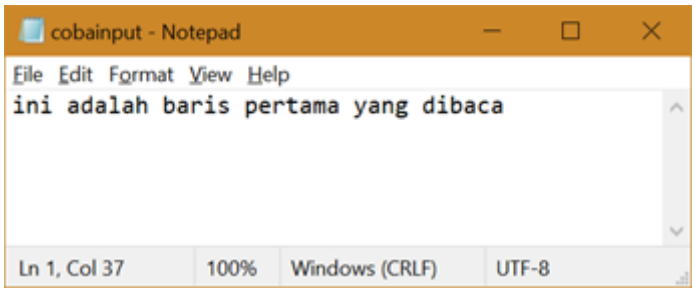
Pada gambar di atas, dengan menggunakan *class* `BufferedReader` terdapat sebuah *file* yang akan dibaca oleh program dengan menggunakan method `FileReader`. Setelah *file* dibaca maka dapat dilanjutkan untuk menuliskannya ke *file* lain dengan method `FileWriter`.

Penggunaan `BufferedInputStream` akan menampung *input* terlebih dahulu pada program, baru akan diteruskan ke *file*, sehingga data yang akan dibaca dapat dilakukan secara per baris. `BufferedInputStream` mendukung method `mark()` dan juga `reset()`. Method `mark()` digunakan untuk menandakan batas data yang akan ditampung, jika data melebihi ukuran mark yang diberikan, maka akan terjadi error. Method `reset()` digunakan untuk mengembalikan atau mengatur ulang `stream`. Berikut ini adalah sintaks umum untuk deklarasi `BufferedInputStream`:

```
FileReader variabel_input = new FileReader(fileName:"file_path");
BufferedReader variabel_buffer = new BufferedReader(variabel_input);
```

Untuk menggunakan `BufferedInputStream`, `FileReader` dideklarasikan terlebih dahulu ke dalam objek baru dengan operator `new`. Dalam `FileReader`, dideklarasikan juga direktori *file* yang akan dibaca. Setelah itu deklarasikan `BufferedReader` ke dalam objek baru dan lakukan pemanggilan `variabel_input` dari `FileReader`. Berikut ini adalah contoh implementasi dari `BufferedInputStream`:

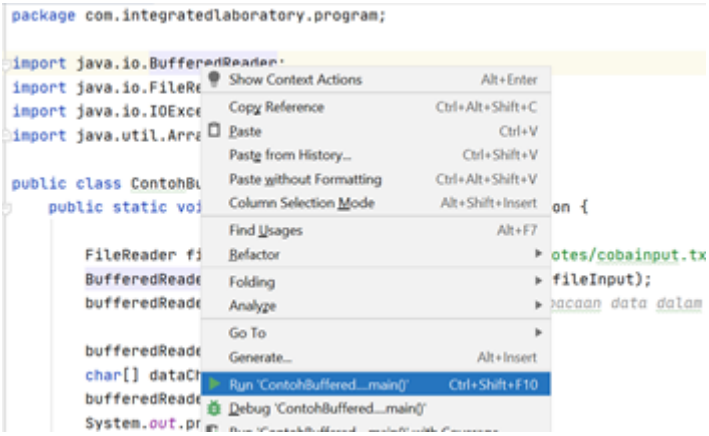
Sebelumnya, buat sebuah *file* dengan nama cobainput.txt pada teks editor:



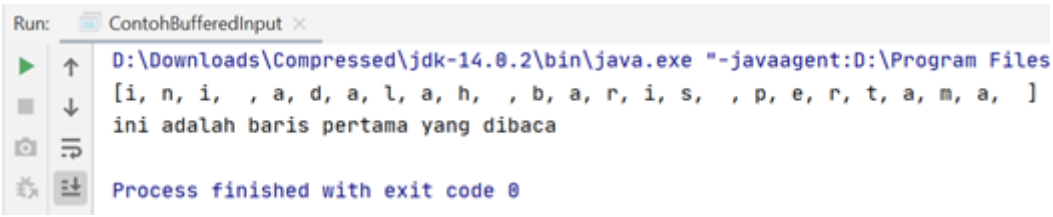

```
package com.integratedlaboratory.program;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;

public class ContohBufferedInput {
    public static void main(String[] args) throws IOException {

        FileReader fileInput = new FileReader("D:/Notes/cobainput.txt");
        BufferedReader bufferedReader = new BufferedReader(fileInput);
        bufferedReader.mark(200); // batas pembacaan data dalam file 200

        bufferedReader.reset();
        char[] dataChar = new char[25];
        bufferedReader.read(dataChar, 0, 25);
        System.out.println(Arrays.toString(dataChar));

        bufferedReader.reset();
        System.out.println(bufferedReader.readLine());
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:



Pada program di atas, `FileReader` dideklarasikan pada objek baru dengan direktori *file* yang akan dibaca yaitu D:/Notes/cobainput.txt. Setelah itu dideklarasikan `BufferedReader` dan memanggil variabel dari `FileReader` yaitu `fileInput`. Lalu dideklarasikan method `mark()` dengan batasan membaca 200. Pada program di atas, dilakukan dua kali pembacaan dalam *file*. Pertama, dalam bentuk array yang dikonversikan ke dalam Character. Kedua, membaca baris dengan jenis karakter.

13.4 BUFFEREDOUTPUTSTREAM

`BufferedOutputStream` merupakan *subclass* dari *class* `java.io` yang akan menulis karakter yang telah ditampung dari sebuah `OutputStream` dengan menggunakan method `FileWriter`. Pada `BufferedOutputStream`, karakter yang ingin ditampilkan menggunakan method `flush()` yang akan menampung data yang ada terlebih dahulu, lalu akan dikirimkan ke *file* lain. Method `flush()` diharuskan ada jika *file* akan dihubungkan dengan *file* lainnya. Berikut ini merupakan sintaks umum dari `BufferedOutputStream`:

```
FileWriter variabel_output = new FileWriter(fileName:"file_path");
BufferedWriter variabel_buffer = new BufferedWriter(variabel_output);
```

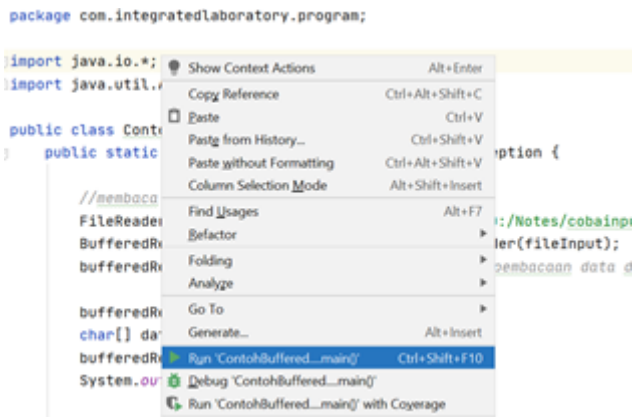
Untuk menggunakan `BufferedOutputStream`, `FileWriter` dideklarasikan terlebih dahulu ke dalam objek baru dengan operator new. Dalam `FileWriter`, dideklarasikan juga direktori *file* yang akan dibaca. Setelah itu deklarasikan `BufferedWriter` ke dalam objek baru dengan nama `variabel_buffer` yang berbeda dari `variabel_buffer` yang terdapat pada `BufferedInputStream` dan lakukan pemanggilan `variabel_output` dari `FileWriter`. Berikut ini adalah contoh implementasi dari `BufferedOutputStream`:

```
package com.integratedlaboratory.program;

import java.io.*;
import java.util.Arrays;

public class ContohBufferedOutput {
```

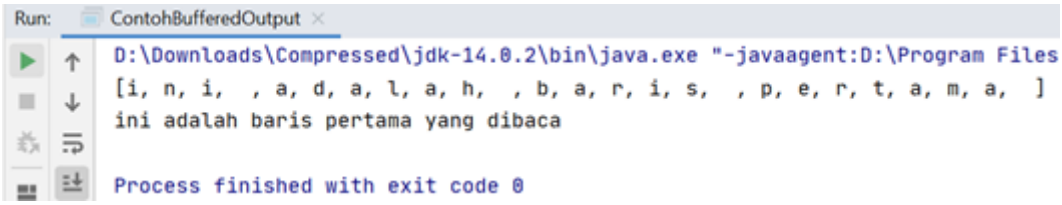
```
    public static void main(String[] args) throws IOException {

        //membaca file
        FileReader fileInput = new FileReader("D:/Notes/cobainput.txt");
        BufferedReader bufferedReader = new BufferedReader(fileInput);
        bufferedReader.mark(200); // batas pembacaan data dalam file 200

        bufferedReader.reset();
        char[] dataChar = new char[25];
        bufferedReader.read(dataChar, 0, 25);
        System.out.println(Arrays.toString(dataChar));

        bufferedReader.reset();
        System.out.println(bufferedReader.readLine());

        //menulis file
        FileWriter fileOutput = new FileWriter("D:/Notes/cobaoutput.txt");
        BufferedWriter bufferedWriter = new BufferedWriter(fileOutput);

        bufferedReader.reset();
        String baris1 = bufferedReader.readLine();

        bufferedWriter.write(baris1,0,baris1.length());
        bufferedWriter.flush();
    }
}
```

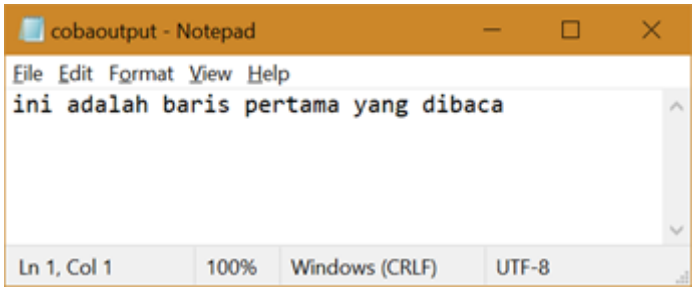
Perintah:

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:





Program di atas melanjutkan dari program pada `BufferedInputStream`. Setelah membaca data yang terdapat pada *file* cobainput.txt, dilanjutkan dengan menulis kembali data yang terdapat pada cobainput.txt ke dalam cobaoutput.txt. *File* cobaoutput.txt tidak perlu dibuat dahulu, karena akan otomatis terbuat ketika mendeklarasikan `FileWriter`.

REFERENSI:

[1] Kasyap, Krishna. 2019. “What is the use of FileInputStream and FileOutputStream in classes in Java?”, https://www.tutorialspoint.com/what-is-the-use-of-fileinputstream-and-fileoutputstream-in-classes-in-java, diakses pada 11 Agustus 2020.

[2] URL: https://www.javatpoint.com/java-fileinputstream-class, diakses pada 11 Agustus 2020.

[3] URL: https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/, diakses pada 11 Agustus 2020.

[4] URL: https://www.youtube.com/watch?v=0suOOClnvaE&list=PLZS-MHyEIRo51w0Hmqi0C8h2KWNzDfo6F&index=63, diakses pada 11 Agustus 2020.

[5] Muhardian, Ahmad. 2015. “Belajar Java: Cara Mengambil Input dan Menampilkan Output”, https://www.petanikode.com/java-input-output/, diakses pada 11 Agustus 2020.

[6] URL: https://www.javatpoint.com/java-fileoutputstream-class, diakses pada 12 Agustus 2020.

[7] URL: https://www.youtube.com/watch?v=KQwIcmm8XzY&list=PLZS-MHyEIRo51w0Hmqi0C8h2KWNzDfo6F&index=66, diakses pada 12 Agustus 2020.

[8] URL: https://www.geeksforgeeks.org/java-io-bufferedinputstream-class-java/, diakses pada 12 Agustus 2020.

[9] Informatika, Kelas. 2020. “BufferedReader dan InputStreamReader pada Java”, https://www.kelasinfor.net/2020/04/bufferedreader-dan-inputstreamreader-java.html, diakses pada 12 Agustus 2020.