# BAB 10
# OPERASI *FILE*

**Tujuan**

1. Memberikan pemahaman kepada mahasiswa tentang operasi *file* di Java.

**Ringkasan Materi**

Data yang diolah di dalam program bersifat sementara dan akan hilang saat program ditutup. Untuk menyimpan data yang dibuat oleh program, data tersebut harus disimpan di dalam sebuah *file*. Data tersebut kemudian akan dapat dibaca dan dipindah ke tempat yang lain, dibaca oleh program yang lain, dst.

Setiap *file* disimpan di dalam suatu direktori. Suatu *file* memiliki nama absolut yang berisi nama *drive*, direktori, nama *file* itu sendiri, hingga ekstensinya. Contohnya, "C:\Data\Java\teks.txt" adalah nama absolut dari *file* "teks.txt" yang berada di dalam direktori/*path* "C:\Data\Java" pada sistem operasi Windows. Sistem operasi yang lain memiliki format penamaan yang berbeda. Contohnya, pada sistem operasi berbasis UNIX, nama absolut dari file yang sama adalah "/Data/Java/teks.txt". Perlu diperhatikan pula bahwa pada sistem operasi berbasis UNIX, nama *file* bersifat *case sensitive*, sedangkan pada Windows tidak.

Operasi *file* termasuk di dalam kategori input-*output* (I/O) sehingga diletakkan pada *package* java.io yang merupakan package input-*output* standar di Java. Java menyediakan sekumpulan *class* baru yang dinamakan Java NIO (new IO) dan diletakkan pada package java.nio. *Package-package* tersebut juga menyediakan operasi-operasi input-*output* untuk jaringan.

Operasi input-*output* berpotensi untuk memunculkan *error*. Oleh karena itu, terdapat kelas *exception* khusus untuk input-*output*, yaitu java.io.IOException. *File* merupakan suatu *resource* yang dapat dibuka dan ditutup. Oleh karena itu, kita bisa memanfaatkan fitur *try-with-resource* yang akan akan menutup *resource* secara otomatis setelah operasi terhadap *resource* tersebut selesai. Hal ini penting, terutama pada komputer dengan kapasitas memori yang terbatas dan jika *file* yang digunakan berukuran besar. Untuk kemudahan implementasi, kita bisa menggunakan *library* Apache Commons IO (https://commons.apache.org/io) yang menyediakan *method-method* untuk operasi-operasi input-*output* yang umum.

**1. Kelas File**

Kelas File adalah kelas yang dapat merepresentasikan *file* dan direktori. Kelas ini dapat digunakan untuk membuat, mendapatkan properti, mengubah nama (*rename*), dan menghapus suatu *file*/direktori. Namun, kelas ini saja tidak dapat digunakan untuk membaca dan menulis ke *file*. Membuat suatu objek kelas File tidak akan secara langsung membuat *file* baru, melainkan hanya sebuah penunjuk/*pointer* ke suatu *file*/direktori saja. Untuk membaca dan menulis ke suatu *file*, dibutuhkan beberapa kelas yang lain. Untuk mengecek apakah suatu objek File merepresentasikan *file* atau direktori, kita dapat menggunakan method `isDirectory()` dan `isFile()`. Tabel 1 menampilkan beberapa *method* yang terdapat pada kelas File.

**Tabel 1.** Beberapa method yang ada pada kelas File.

| No. | Method | Deskripsi |
|---|---|---|

| | | |
|---|---|---|
| 1 | `canExecute()` | Mengecek apakah suatu *file* dapat dieksekusi. |
| 2 | `canRead()` | Mengecek apakah suatu *file* dapat dibaca. |

| | | |
|---|---|---|
| 3 | `canWrite()` | Mengecek apakah suatu *file* dapat ditulis. |
| 4 | `createNewFile()` | Membuat *file* kosong baru. |
| 5 | `delete()` | Menghapus *file*/direktori. |
| 6 | `deleteOnExit()` | Menghapus *file*/direktori setelah eksekusi program berakhir. |
| 7 | `equals(Object o)` | Mengecek kesamaan dengan objek o. |
| 8 | `exists()` | Mengecek keberadaan *file*/direktori. |
| 10 | `getAbsolutePath()` | Mendapatkan nama absolut suatu *file*/direktori. |
| 11 | `getFreeSpace()` | Mendapatkan *free space* dari *drive*/partisi. |
| 12 | `getName()` | Mendapatkan nama *file*/direktori. |
| 13 | `getParent()` | Mendapatkan nama *parent* dari *file*/direktori sebagai *string*. |
| 14 | `getParentFile()` | Mendapatkan *parent* dari *file*/direktori sebagai objek File. |
| 16 | `getTotalSpace()` | Mendapatkan ukuran dari *drive*/partisi. |
| 17 | `getUsableSpace()` | Mendapatkan ukuran yang dapat digunakan dari *drive*/partisi. |
| 18 | `isAbsolute()` | Mengecek apakah nama *file*/direktori bersifat absolut. |
| 19 | `isDirectory()` | Mengecek apakah objek berupa direktori. |
| 20 | `isFile()` | Mengecek apakah objek berupa *file*. |
| 21 | `isHidden()` | Mengecek apakah *file*/direktori berstatus *hidden*. |
| 22 | `lastModified()` | Mendapatkan waktu terakhir *file*/direktori dimodifikasi. |
| 23 | `length()` | Mendapatkan ukuran *file*/direktori dalam satuan *bytes*. |
| 24 | `list()` | Mendapatkan daftar *file*/direktori di dalam suatu direktori. |

| | | |
|---|---|---|
| 25 | `listFiles()` | Mendapatkan daftar *file*/direktori di dalam suatu direktori sebagai objek File. |
| 26 | `mkdir()` | Membuat direktori baru. |
| 27 | `mkdirs()` | Membuat beberapa direktori baru. |
| 28 | `renameTo()` | Mengubah nama *file*/direktori. |
| 29 | `setExecutable()` | Mengubah status *file* menjadi *executable*. |
| 30 | `setLastModified()` | Mengubah waktu modifikasi terakhir *file*/direktori. |
| 31 | `setReadOnly()` | Mengubah status *file* menjadi *read only*. |
| 32 | `setReadable()` | Mengubah status *file* menjadi *readable*. |
| 33 | `setWritable()` | Mengubah status *file* menjadi *writable*. |

**2. Menulis Teks ke *File***

Terdapat banyak cara dan *class* di Java untuk membaca isi *file*, antara lain `OutputStream`, `ByteArrayOutputStream`, `FileOutputStream`, `RandomAccessFile`, `PipedOutputStream`, `BufferedOutputStream`, `FilterOutputStream`, `DataOutputStream`, `PrintStream`, `ObjectOutputStream`, `FileWriter`, `PipedWriter`, `BufferedWriter`, `FilterWriter`, `StringWriter`, dan `PrintWriter`, dan `Files` (NIO).

**3. Membaca Isi *File***

Terdapat banyak cara dan *class* pula di Java untuk membaca isi *file*, antara lain `InputStream`, `ByteArrayInputStream`, `FileInputStream`, `RandomAccessFile`, `PipedInputStream`, `BufferedInputStream`, `FilterInputStream`, `DataInputStream`, `ObjectInputStream`, `FileReader`, `PipedReader`, `BufferedReader`, `FilterReader`,`Scanner`, `StringReader`, dan `Files` (NIO).

Sebelum melakukan operasi pembacaan isi *file*, sangat disarankan untuk mengecek apakah *file* yang akan dibaca benar-benar ada dan dapat dibaca. Hal ini penting untuk mencegah munculnya *error* jika *file* yang ingin dibaca ternyata tidak ada
atau tidak dapat dibaca. Pengecekan ini dapat dilakukan dengan *method* `exists()` dan `canRead()`.

**4. Mengubah Nama dan Menghapus *File***

Mengubah nama *file*/direktori dilakukan menggunakan *method* `renameTo()`,

sedangkan menghapus *file*/direktori dilakukan menggunakan *method* `delete()` atau `deleteOnExit()`. Sama halnya dengan membaca *file*, sangat disarankan untuk mengecek keberadaan *file* yang akan diubah namanya atau dihapus. Method `renameTo()` dan `delete()` mengembalikan nilai Boolean yang mengindikasikan apakah operasi berhasil atau tidak. Untuk operasi hapus, jika yang akan dihapus adalah sebuah direktori, maka direktori tersebut harus dalam keadaan kosong. Jika tidak, maka operasi akan gagal.

**Pelaksanaan Percobaan**

**A. Menulis teks ke *file***

Ketikkan kode program di bawah ini dan analisis *output* dari program tersebut!

TulisFile1.java

```
1  import java.io.FileWriter;
2  import java.io.IOException;
3  import java.util.Scanner;
4
5  public class TulisFile {
6
7   public static void main(String[] args) {
8   var keyboard = new Scanner(System.in);
9
10   System.out.print("Masukkan teks yang akan disimpan: ");
11 var text = keyboard.nextLine();
12
13   try (var writer = new FileWriter("test.txt", false)) {
14 writer.write(text);
15   } catch (IOException e) {
16   System.err.println("Gagal menulis ke file");  }
17   }
18  }
19
20
```

TulisFile2.java

```
1  import java.io.IOException;
2  import java.nio.file.Files;
3  import java.nio.file.Paths;
4  import java.util.Scanner;
5
6  public class TulisFile2 {
7
8   public static void main(String[] args) {
9   var keyboard = new Scanner(System.in);
10  var path = Paths.get("test.txt");
11
```

```
12
13   System.out.print("Masukkan teks yang akan disimpan: ");
     var text = keyboard.nextLine();
```

```
14   try {
15   Files.writeString(path, text);
16   } catch (IOException e) {
17   System.err.println("Gagal menulis ke file");  }
18   }
19  }
20
21
```

TulisFile3.java

```
 1  import java.io.IOException;
 2  import java.io.PrintWriter;
 3  import java.util.Scanner;
 4
 5  public class TulisFile3 {
 6
 7   public static void main(String[] args) {
 8   var keyboard = new Scanner(System.in);
 9
10   System.out.print("Masukkan teks yang akan disimpan: ");
11  var text = keyboard.nextLine();
12
13   try (var writer = new PrintWriter("test.txt")) {
14  writer.println(text);
15   } catch (IOException e) {
16   System.err.println("Gagal menulis ke file");  }
17   }
18  }
19
```

**B. Membaca isi file teks**

Ketikkan kode program di bawah ini dan analisis *output* dari program tersebut!

BacaFile1.java

```
1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class BacaFile1 {
7
8   public static void main(String[] args) {
9   var filename = "test.txt";
10  var file = new File(filename);
11
12  if (file.exists() && file.canRead()) {
13
14  try (var reader = new FileReader(filename);  var
15 buffer = new BufferedReader(reader)) {  var line =
16 buffer.readLine();
17
18  while (line != null) {
19  System.out.println(line);
20                  line = buffer.readLine();
```

```
21   }
22   } catch (IOException e) {
23   System.err.println("Gagal membaca file");  }
24   } else {
25   System.out.println("File tidak ada atau tidak bisa
26 dibaca");
      }
27   }
28 }
29
```

BacaFile2.java

```
1  import java.io.File;
2  import java.io.FileReader;
3  import java.io.IOException;
4  import java.util.Scanner;
5
6  public class BacaFile2 {
7
8   public static void main(String[] args) {
9   var filename = "test.txt";
10  var file = new File(filename);
11
12  if (file.exists() && file.canRead()) {
13
14  try (var reader = new FileReader(filename);  var
15 scanner = new Scanner(reader)) {  while
16 (scanner.hasNextLine()) {
17  System.out.println(scanner.nextLine());  }
18  } catch (IOException e) {
19  System.err.println("Gagal membaca file");  }
20  } else {
21  System.out.println("File tidak ada atau tidak bisa
22 dibaca");
23  }
    }
24 }
25
26
```

BacaFile3.java

```
1  import java.io.IOException;
2  import java.nio.file.Files;
3  import java.nio.file.Paths;
4  import java.util.List;
5
6  public class BacaFile3 {
7
8   public static void main(String[] args) {
9   var path = Paths.get("test.txt");
10  var file = path.toFile();
11
12  if (file.exists() && file.canRead()) {
```

```
13   List<String> text = null;
14
15   try {
16   text = Files.readAllLines(path);
17   } catch (IOException e) {
18   System.err.println("Gagal membaca file");  }
19
20   for (var line : text) {
21   System.out.println(line);
22   }
23   } else {
24   System.out.println("File tidak ada atau tidak bisa
25  dibaca");
      }
26   }
27  }
28
```

**C. Informasi tentang *file***

Ketikkan kode program di bawah ini dan analisis *output* dari program tersebut!

FileInfo.java

```
1  import java.io.File;
2  import java.util.Date;
3
4  public class FileInfo {
5
6   public static void main(String[] args) {
7   var file = new File("test.txt");
8
9   System.out.println("File ada? " + file.exists());
10 System.out.printf("Path: %s\n",
   file.getAbsolutePath());
11  System.out.println("Last modified: " + new
   Date(file.lastModified()));
12  System.out.printf("Ukuran file: %d bytes\n",
   file.length());
13  System.out.println("File bisa dibaca? " +
   file.canRead());
14  System.out.println("File bisa ditulis? " +
   file.canWrite());
15  System.out.println("Apakah sebuah direktori? " +
   file.isDirectory());
16  System.out.println("Apakah sebuah file? " +
   file.isFile());
17  System.out.println("Apakah hidden? " +
18 file.isHidden());
    }
19
```

```
20  }
```

**D. Mengubah Nama *File***

Ketikkan kode program di bawah ini dan analisis *output* dari program tersebut!

Rename.java

```
 1  import java.io.File;
 2
 3  public class Rename {
 4
 5   public static void main(String[] args) {
 6   var file = new File("test.txt");
 7   var fileBaru = new File("test-baru.txt");
 8
 9   if (file.exists() && !fileBaru.exists()) {
10  var result = file.renameTo(fileBaru);
11
12   System.out.println((result ? "Berhasil" : "Gagal")  + "
    mengubah nama file");
13   } else {
14   System.out.println("File tidak ada atau file dengan  nama
    yang baru sudah ada");
15   }
16   }
17  }
```

**E. Menghapus File**

Ketikkan kode program di bawah ini dan analisis *output* dari program tersebut!

Delete.java

```
 1  import java.io.File;
 2
 3  public class Delete {
 4
 5   public static void main(String[] args) {
 6   var file = new File("test-baru.txt");
 7
 8   if (file.exists()) {
 9   var result = file.delete();
10
11   System.out.println((result ? "Berhasil" : "Gagal")  + "
12  menghapus file");
    } else {
13   System.out.println("File tidak ada");  }
14   }
15  }
```

| | |
|---|---|
| 16 | |

**Tugas Praktikum**

1. Jalankan kode TulisFile1.java beberapa kali dan amati yang terjadi pada file yang ditulis. Kemudian, Pada baris 13, ubah parameter `false` menjadi `true`. Kemudian jalankan kode tersebut beberapa kali dan amati yang terjadi pada *file* yang ditulis.
   **Jawaban:**
   - Saat false (kode asli): Ketika TulisFile1.java dijalankan beberapa kali dengan new FileWriter("test.txt", false), isi dari test.txt akan ditimpa (di-overwrite) dengan teks masukan yang baru setiap kali dijalankan. Parameter false menunjukkan bahwa file harus dikosongkan terlebih dahulu sebelum menulis jika file tersebut sudah ada, atau membuat file baru jika belum ada.
   - Saat true (kode yang dimodifikasi): Ketika TulisFile1.java dijalankan beberapa kali dengan new FileWriter("test.txt", true), teks masukan yang baru akan ditambahkan (di-append) ke akhir test.txt. Parameter true menunjukkan bahwa data baru harus ditambahkan ke akhir file tanpa menghapus konten yang sudah ada.

2. Buat kode program untuk mendapatkan ukuran *file* dalam satuan KB jika ukuran *file* tersebut < 1 MB dan dalam satuan MB jika ukuran *file* tersebut >= 1 MB.
   **Jawaban:**
   Untuk mencapai ini, kita bisa menggunakan metode length() dari kelas File untuk mendapatkan ukuran file dalam bytes. Kemudian, kita bisa mengkonversinya ke KB atau MB berdasarkan kondisi yang diberikan.

3. Buat kode program untuk menampilkan nama dari semua *file* yang ada di dalam suatu direktori. Petunjuk: gunakan perulangan dan *method* `list()` atau `listFiles()`.
   **Jawaban:**
   Metode list() dari kelas File mengembalikan array string yang berisi nama-nama file dan direktori di dalam direktori. Untuk tugas ini, list() sudah cukup untuk mendapatkan hanya namanya.

4. Buat kode program untuk menghapus suatu direktori beserta semua *file* yang ada di dalamnya. Asumsi: di dalam direktori tersebut, hanya ada *file-file* saja, tidak ada subdirektori.
   **Jawaban:**
   Untuk menghapus direktori beserta file-filenya, pertama, hapus semua file di dalam direktori, lalu hapus direktori itu sendiri. Metode listFiles() akan sangat berguna di sini.

5. Apakah yang salah dengan *statement* berikut? Berikan penjelasan.
   `var file = new File(“C:\Data\Java\teks.txt”);`
   **Jawaban:**
   Kesalahan pada pernyataan var file = new File("C:\Data\Java\teks.txt"); terletak pada penggunaan karakter backslash (\). Dalam Java (dan banyak bahasa pemrograman lainnya), backslash adalah karakter escape. Ini berarti digunakan untuk memperkenalkan urutan karakter khusus (misalnya, \n untuk baris baru, \t untuk tab).

   Untuk merepresentasikan backslash literal dalam sebuah string,perlu meng-escape-nya dengan menggunakan double backslash (\\).

   Pernyataan yang Benar:

   **var file = new File("C:\\Data\\Java\\teks.txt");**

   Atau, terutama untuk path Windows, bisa menggunakan forward slash (/), yang biasanya ditangani dengan benar oleh kelas File Java di berbagai sistem operasi.

   **var file = new File("C:/Data/Java/teks.txt");**

6. Apa yang akan terjadi jika kita mencoba untuk membaca isi dari suatu *file* tetapi *file* tersebut tidak ada dan kita tidak melakukan pengecekan lebih dulu?
   **Jawaban:**
   Jika mencoba membaca isi dari file yang tidak ada tanpa memeriksa keberadaannya terlebih dahulu, **java.io.FileNotFoundException** (yang merupakan subkelas dari IOException) akan thrown pada saat eksekusi (runtime). Ini karena operasi input-output file berpotensi menimbulkan kesalahan, dan mencoba mengakses file yang tidak ada adalah skenario umum untuk kesalahan semacam itu.
   Sangat disarankan untuk memeriksa apakah file ada dan dapat dibaca menggunakan metode seperti exists() dan canRead() sebelum mencoba membaca isinya untuk mencegah exception saat runtime tersebut.