Pemrograman Berorientasi Obyek

Pertemuan 9: File dan Exception Handling pada Java

Tambahan Materi Java

- File Handling
- Exception Handling

Operasi File

- Dalam pemrograman skala menengah dan besar, input dan output data dalam program sering berjumlah banyak dan perlu ditulis/disimpan dalam sebuah file
- Pada Java, operasi file termasuk dalam package besar Java Input/Output (`java.io`), yang juga mencakup membaca dan menampilkan data dari segala sumber (file, keyboard, network)
- Beberapa macam operasi file mencakup:
  - Membuat file
  - Menghapus file
  - Membaca file
  - Menulis file
  - Mengubah hak akses file

1. Membuat File

- Menggunakan method `createNewFile()` dari class `File`
- Method mengembalikan nilai `true` jika file berhasil dibuat atau `false` jika gagal

```
import java.io.File;
import java.io.IOException;

public class FileHandling {

	public static void main(String[] args) {
		File file = new File("data.txt");
		try {
			boolean createNewFile = file.createNewFile();
			System.out.println("File Created = "+createNewFile);
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

2. Menghapus File

- Menggunakan method `delete()` dari class `File`
- Method mengembalikan nilai `true` jika file berhasil dihapus atau `false` jika gagal

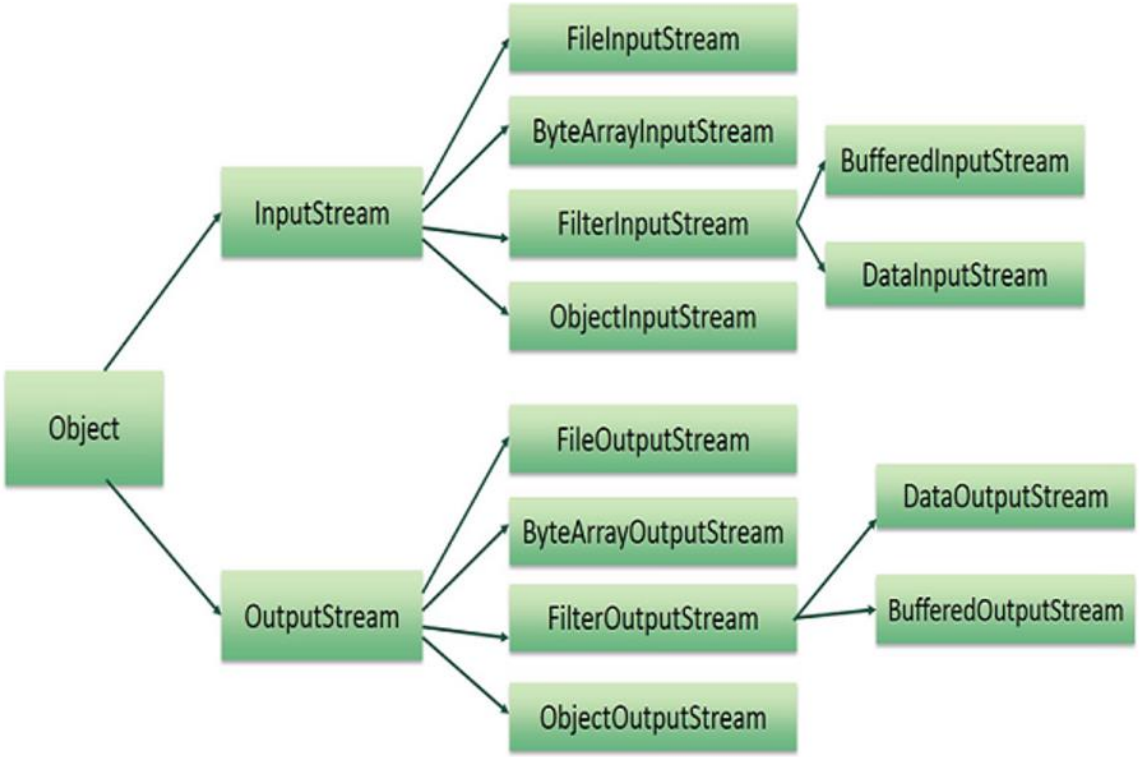
```
import java.io.File;

public class FileHandling {

        public static void main(String[] args) {
                File file = new File("data.txt");
                boolean delete = file.delete();
                System.out.println("File deleted = " + delete);
        }
}
```

Stream

- Stream adalah arus/deretan data
- **InputStream** digunakan untuk membaca data dari sebuah sumber dan **OutputStream** digunakan untuk menulis data ke sebuah tujuan



Hirarki class Input dan Output Stream

3. Membaca File

- Beberapa cara untuk membaca file dalam Java dengan menggunakan Class `BufferedReader`, `FileReader`, atau `File`
- Berikut contoh code untuk membaca file baris per baris:

```
1  package file;
2
3  import java.io.*;
4
5  public class ReadFile {
6      public static void main(String[] args){
7          File file=new File("src/file/data.txt");
8          try {
9              FileInputStream fis=new FileInputStream(file);
10             InputStreamReader isr=new InputStreamReader(fis);
11             BufferedReader br=new BufferedReader(isr);
12
13             String line;
14             try {
15                 while((line=br.readLine())!=null){
16                     System.out.println(line);
17                 }
18                 br.close();
19             } catch (IOException e) {
20                 // TODO Auto-generated catch block
21                 e.printStackTrace();
22             }
23         } catch (FileNotFoundException e) {
24             // TODO Auto-generated catch block
25             e.printStackTrace();
26         }
27     }
28 }
```

FileInputStream: mengambil byte input dari file di file system

InputStreamReader: jembatan antara byte stream dengan character stream

BufferedReader: Membaca teks dari character-input stream dan mem-buffernya agar proses pembacaan karakter, array, dan baris lebih efisien

3. Membaca file

- Ada banyak cara lain untuk membaca file teks di dalam Java
- Lebih lengkapnya dapat melihat tutorial di http://www.journaldev.com/867/java-read-text-file

4. Menulis File

- Beberapa cara untuk menulis file: dengan menggunakan class `FIleWriter`, `BufferedWriter`, `Files`, atau `FileOutputStream`
- Berikut contoh code untuk menulis ke file dengan menggunakan buffer:

```
1  package file;
2  import java.io.*;
3
4  public class BufferedWriteFile {
5      public static void main(String[] args){
6          try {
7              FileOutputStream fos=new FileOutputStream("src/file/testout.txt");
8              BufferedOutputStream bos = new BufferedOutputStream(fos);
9              String s="Welcome to Java File Handling";
10             byte b[]=s.getBytes();
11             try {
12                 bos.write(b);
13                 bos.flush();
14                 bos.close();
15                 fos.close();
16             } catch (IOException e) {
17                 // TODO Auto-generated catch block
18                 e.printStackTrace();
19             }
20         } catch (FileNotFoundException e) {
21             // TODO Auto-generated catch block
22             e.printStackTrace();
23         }
24     }
25 }
```

4. Menulis File

- Ada banyak cara lain untuk menulis file teks di dalam Java
- Lebih lengkapnya dapat melihat tutorial di http://www.journaldev.com/878/java-write-to-file

5. Mengubah hak akses file

- Sebelum Java 7 (versi 1.6 dan di bawahnya), setting hak akses hanya dapat dibedakan menjadi 2: owner dan selain owner
- Mulai Java 7 (versi 1.7 ke atas), Java memiliki class `PosixFilePermission` pada package `java.nio.*` untuk lebih memudahkan setting hak akses pada file dan memungkinkan untuk membagi hak akses menjadi 3: owner, grup dari owner, dan lainnya

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.attribute.PosixFilePermission;
import java.util.HashSet;
import java.util.Set;

public class FilePermissions {

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

      //using PosixFilePermission to set file permissions 777
      Set<PosixFilePermission> perms = new HashSet<PosixFilePermission>();
      //add owners permission
      perms.add(PosixFilePermission.OWNER_READ);
      perms.add(PosixFilePermission.OWNER_WRITE);
      perms.add(PosixFilePermission.OWNER_EXECUTE);
      //add group permissions
      perms.add(PosixFilePermission.GROUP_READ);
      perms.add(PosixFilePermission.GROUP_WRITE);
      perms.add(PosixFilePermission.GROUP_EXECUTE);
      //add others permissions
      perms.add(PosixFilePermission.OTHERS_READ);
      perms.add(PosixFilePermission.OTHERS_WRITE);
      perms.add(PosixFilePermission.OTHERS_EXECUTE);

      Files.setPosixFilePermissions(Paths.get("/tmp/file.txt"), perms);
   }
}
```

Output:

```
$ ls -l /tmp/file.txt
-r--r--r-- 1 dennis dennis 0 Jan 25 16:44 /tmp/file.txt

$ java FilePermissions

$ ls -l /tmp/file.txt
-rwxrwxrwx 1 dennis dennis 0 Jan 25 16:44 /tmp/file.txt
```

Sebelum dilakukan setting file permission 777

Setelah dilakukan setting file permission 777

Exception Handling

- Mengapa perlu exception?
  - Program komputer tidak selalu melakukan apa yang diharapkan oleh programmernya
  - Pada suatu waktu, sebuah program dapat saja gagal dijalankan, karena kesalahan programmer atau penyebab eksternal
    - File dapat terhapus secara tidak sengaja sehingga program kita tidak dapat membukanya, atau kita mencoba membuka file yang salah
    - Koneksi jaringan dapat terputus dan program kita dapat terputus hubungan dengan komputer lain di Internet
  - Apapun jenis kegagalannya, untuk menangani kondisi tak terduga seperti ini, kita dapat menggunakan fitur Exceptions

Exception

- Exception adalah variabel yang akan “dibangkitkan” atau “dijalankan” ketika terjadi suatu kesalahan
- Jika terjadi exception, eksekusi program akan terhenti dan interpreter Java akan mencari bagian code yang dapat “menangkap” atau “menangani” exception
  - Contoh: Jika sebuah file tidak sengaja terhapus, dan di dalam program memanggil perintah `FileInputStream fis=new FileInputStream(new File(“src/file/data.txt”)` untuk membuka file tersebut, maka exception akan terjadi
- Interpreter Java mengetahui apa masalahnya: File yang dicari tidak ada, tetapi ia tidak tahu bagaimana menangani masalah ini. Apakah programnya langsung berhenti dan keluar? Ataukah mencoba membuka file yang lain?
- Inilah yang dilakukan oleh bagian program yang bertugas untuk “menangkap” exception, yang disebut ***exception handler***

Exception pada Java

- Ada 2 macam exception:
  - Checked exception:
    - exception yang terjadi saat *compile time*
    - Tidak boleh diabaikan, programmer harus menangani exception ini
    - Contoh: class FileReader memiliki exception jika file yang akan dibaca ternyata tidak ada, maka muncul *FileNotFoundException*

```
import java.io.File;
import java.io.FileReader;

public class FilenotFound_Demo {

   public static void main(String args[]) {
      File file = new File("E://file.txt");
      FileReader fr = new FileReader(file);
   }
}
```

Catatan: Method `read()` dan `write()` melemparkan IOException, maka ini juga perlu ditangani

Output:

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must be caught
      FileReader fr = new FileReader(file);
                      ^
1 error
```

Exception pada Java

- Unchecked exception:
  - Exception yang terjadi saat *run time,* termasuk bug pada program
  - Contoh:
    - Jika kita mendeklarasikan array berukuran 5 pada program, lalu berusaha memanggil/mengakses elemen ke-6, maka muncul exception ***ArrayIndexOutOfBoundsException***

```
public class Unchecked_Demo {

   public static void main(String args[]) {
      int num[] = {1, 2, 3, 4};
      System.out.println(num[5]);
   }
}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

Menangkap Exception

- Sebuah method dapat menangkap exception dengan menggunakan kombinasi kata kunci **try** dan **catch**
- Block try/catch ditempatkan pada bagian code yang dapat menghasilkan exception. Dengan kata lain, code tersebut **diproteksi**
- Sintaks:

```
try {
   // Protected code
}catch(ExceptionName e1) {
   // Catch block
}
```

Contoh code

- Berikut contoh program yang mendeklarasikan array dengan 2 elemen, lalu mencoba mengakses elemen ke-3 yang akan menghasilkan exception

```
// File Name : ExcepTest.java
import java.io.*;

public class ExcepTest {

   public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

Output:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple Catch Blocks

- Multiple exception dapat terjadi dalam sebuah program, sehingga diperlukan pula penulisan multiple blok try/catch.
- Contoh:

```
try {
   file = new FileInputStream(fileName);
   x = (byte) file.read();
}catch(IOException i) {
   i.printStackTrace();
   return -1;
}catch(FileNotFoundException f) // Not valid! {
   f.printStackTrace();
   return -1;
}
```

Catatan:
Mulai Java 7, kita dapat menangani lebih dari 1 exception dengan block catch tunggal

```
catch (IOException|FileNotFoundException ex) {
   logger.log(ex);
   throw ex;
```