

---

# Pengenalan Java IO

---

# Pengenalan Java IO

- Java merupakan teknologi yang sangat lengkap, salah satu nya untuk melakukan proses IO (Input Output)
- Java IO merupakan fitur di Java yang bisa digunakan untuk berinteraksi dengan data diluar aplikasi kita, yaitu dari disk atau file system
- Dengan Java IO, kita bisa dengan mudah membaca dan menyimpan data di file

---

## Kenapa di File?

- Saat kita menjalankan aplikasi Java, dia akan berjalan di RAM (memory), artinya saat aplikasi selesai dijalankan, maka seluruh data akan hilang. Dan ketika kita menjalankan ulang lagi datanya, secara otomatis akan mulai dari awal lagi
- File bisa digunakan untuk menyimpan data pada aplikasi Java kita. Dengan menyimpan data di file, saat aplikasi dihentikan, data akan tetap akan karena disimpan di file di disk / file system.

---

## Library Java IO

- Java IO merupakan fitur bawaan dari Java, oleh karena itu secara otomatis kita bisa menggunakan fitur ini tanpa menambahkan library atau dependency external apapun
- Kebanyakan fitur Java IO terdapat di package `java.io`
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/package-summary.html>

---

# Konsep Java IO

---

## Konsep Java IO

- Di dalam Java IO, terdapat dua konsep utama yang wajib kita ketahui
- Pertama adalah lokasi sumber data (resource) yang perlu diakses, dan
- Kedua adalah membuka stream (untuk membaca atau menulis) terhadap sumber data (resource) tersebut



# Resource

- Resource atau sumber data di Java IO bisa menggunakan beberapa mekanisme.
- Menggunakan class File, yang sudah bisa dilakukan sejak versi Java 1.0
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/File.html>
- Dan menggunakan class Path yang bisa digunakan sejak versi Java 7.0
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/Path.html>

---

# IO Stream

- IO Stream merupakan representasi dari input data atau output data menuju resource.
- Stream mendukung banyak jenis data, seperti tipe data primitive, String, Object, dan lain-lain
- IO Stream berbeda dengan Java Stream yang dikenalkan sejak Java 8, walaupun namanya sama, tapi secara fitur dan konsep sangat berbeda
- IO Stream di Java didefinisikan oleh empat Interface utama :
- Stream untuk karakter: Reader dan Writer
- Stream untuk bytes: InputStream dan OutputStream

---

# Membuat Project

---

# Membuat Project

- <https://start.spring.io/>

---

# File



# File

- File adalah cara lama mengakses file, walaupun sekarang direkomendasikan menggunakan Path, namun masih banyak yang menggunakan File
- File direpresentasikan oleh class File
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/File.html>



## Kode : Membuat Object File

```
no usages new *
@Test
void createFile() {
    File file = new File("file.txt");
    Assertions.assertEquals("file.txt", file.getName());
    Assertions.assertFalse(file.exists());
}
```

---

## Absolute vs Relative Path Name

- Saat kita membuat file, terdapat parameter pathname pada constructor nya
- Kita bisa menggunakan Absolute atau Relative Path Name
- Absolute artinya kita sebutkan file dari posisi awal folder, misal di Mac / Linux kita sebutkan misalnya : /Users/khannedy/folder/namafile.txt, atau di Windows kita sebutkan C:\home\khannedy\folder\namafile.txt
- Relative artinya kita sebutkan file dari posisi kita menjalankan aplikasi. Misal jika folder project kita di /Users/khannedy/namaproject. Saat kita menggunakan path namafile.txt, artinya mengakses /Users/khanedy/namaproject/namafile.txt
- Pada Relative, kita bisa gunakan .. untuk naik ke folder diatas nya, misal ../namafile.txt, artinya mengakses file /Users/khannedy/namafile.txt

---

# Path



# Path

- Sejak Java 7, dikenalkan class Path yang bisa digunakan untuk mengakses resource sebagai pengganti class File
- Path adalah sebuah interface, untuk membuat Path, kita bisa gunakan static method Path.of(pathname)
- Path terdapat di package java.nio.file
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/Path.html>

---

## Kode : Membuat Object Path

```
@Test
void createPath() {
    Path path = Path.of("files.txt");

    Assertions.assertEquals("files.txt", path.toString());
    Assertions.assertFalse(path.isAbsolute());
}
```



# Files

- Path tidak memiliki banyak sekali method seperti di class File
- Namun terdapat class Files yang bisa digunakan untuk mendapatkan informasi dari Path
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/Files.html>



## Kode : Menggunakan Files

```
@Test  
void usingFiles() {  
    Path path = Path.of("pom.xml");  
  
    Assertions.assertEquals("pom.xml", path.toString());  
    Assertions.assertFalse(path.isAbsolute());  
    Assertions.assertTrue(Files.exists(path));  
}
```

---

# Manipulasi File

---

## Manipulasi File

- Seperti yang dijelaskan di awal, membuat File atau Path bukan berarti file otomatis terbuat, itu hanya object di Java, bukan representasi nyata di File System
- Namun dengan menggunakan class Files, kita bisa melakukan manipulasi file seperti membuat, menghapus atau membacanya

---

# Manipulasi File

Method	Keterangan
Files.createFile(path)	Membuat file
Files.delete(path) / Files.deleteIfExists(path)	Menghapus file / jika ada
Files.copy(source, target)	Menyalin file
Files.move(source, target)	Memindahkan file

---

## Kode : Manipulasi File

```
no usages - new
@Test
void fileManipulation() throws IOException {
    Path path = Path.of("files.txt");
    Files.createFile(path);
    Assertions.assertTrue(Files.exists(path));

    Files.delete(path);
    Assertions.assertFalse(Files.exists(path));
}
```

---

# Manipulasi Directory

---

# Manipulasi Directory

- Directory atau Folder juga bisa dimanipulasi menggunakan class Files

---

# Manipulasi Directory

Method	Keterangan
Files.createDirectories(path)	Membuat banyak directory jika tidak ada
Files.createDirectory(path)	Membuat satu directory
Files.newDirectoryStream(path)	Membaca semua isi file di directory
Files.isDirectory(path)	Mengecek apakah path sebuah directory



## Kode : Manipulasi Directory

```
@Test
void directoryManipulation() throws IOException {
    Path path = Path.of("contoh");
    Files.createDirectory(path);
    Assertions.assertTrue(Files.isDirectory(path));
    Assertions.assertTrue(Files.exists(path));

    Files.delete(path);
    Assertions.assertFalse(Files.exists(path));
}
```

---

# Closable Interface

---

## Closable Interface

- Semua IO Stream di Java adalah turunan dari interface `java.io.Closable`, dimana terdapat method `close()` untuk menutup Resource yang sudah dibuka.
- Hal ini agar tidak terjadi memory leak (kondisi dimana kita lupa menutup IO Stream sehingga data tidak bisa dihapus di memory oleh Java Garbage Collector)



## Kode : Menutup IO

```
void closeIO() {
    Path path = Path.of("pom.xml");
    InputStream inputStream = null;
    try {
        inputStream = Files.newInputStream(path);
        // do with input stream
    } catch (IOException exception) {
        Assertions.fail(exception);
    } finally {
        if (inputStream != null) {
            try {
                inputStream.close();
            } catch (IOException exception) {
                Assertions.fail(exception);
            }
        }
    }
}
```

---

## Try with Resource

- Seperti yang sudah dibahas di materi Java OOP, Try di Java memiliki fitur yang bisa digunakan untuk menutup Resource Closable secara otomatis
- Dengan begitu, kita tidak perlu lagi melakukan close() IO Resource secara manual lagi seperti pada kode sebelumnya



## Kode : Try with Resource

```
no usage now

@Test
void closeIOWithResource() {
    Path path = Path.of("pom.xml");
    try (InputStream inputStream = Files.newInputStream(path)) {
        // do with input stream
    } catch (IOException exception) {
        Assertions.fail(exception);
    }
}
```

---

# Numeric dan Character



# Numeric Data

- Input/Output Stream di Java membaca dan menulis data dalam tipe data bytes
- Untuk belajar Input dan Output Stream di Java, kita harus mengerti bagaimana Java berurusan dengan tipe data bytes, integers, characters da tipe data primitive lainnya

---

# Integer Data

- Tipe data int adalah 4 byte ukuran
- Tipe data long adalah 8 byte ukuran
- Tipe data short adalah 2 byte ukuran
- Namun walaupun data sebenarnya harus menggunakan tipe data byte, namun karena secara default saat kita membuat number di Java selalu menggunakan tipe data int, maka hampir kebanyakan return value dan parameter di IO Stream di Java menggunakan tipe data int
- Namun pada kasus tertentu yang data bytes banyak, kebanyakan menggunakan parameter atau return value array of bytes



# Character Data

- Selain Input dan Output Stream, di Java juga kita juga bisa membuat Stream khusus untuk jenis Resource Text menggunakan Reader dan Writer
- Dengan menggunakan Reader dan Writer, kita tidak perlu melakukan konversi tipe data Text menjadi Numeric secara manual
- Tipe data Text di Reader dan Writer bisa menggunakan tipe data char, char[] atau String

---

# Menulis File Kecil

---

## Menulis File Kecil

- Pada kasus jika kita ingin menulis file dengan ukuran kecil, kita bisa menggunakan class Files
- Terdapat banyak method yang bisa kita gunakan pada kasus sederhana untuk membuat file dengan ukuran kecil
- Files.write(path, bytes) untuk menulis file dengan data bytes[]
- Files.writeString(path, string) untuk menulis file dengan data String



## Kode : Menulis File Kecil

```
@rest
void writeSmallFile() throws IOException {
    Path path1 = Path.of("small1.txt");
    byte[] bytes = "Hello World".getBytes();
    Files.write(path1, bytes);

    Path path2 = Path.of("small2.txt");
    Files.writeString(path2, "Hello World");
}
```

---

# Membaca File Kecil

---

## Membaca File Kecil

- Pada kasus jika kita ingin membaca file dengan ukuran kecil, kita bisa menggunakan class Files
- Terdapat banyak method yang bisa kita gunakan pada kasus sederhana untuk membaca file dengan ukuran kecil
- Files.readAllBytes(path) untuk membaca file menjadi data bytes[]
- Files.readString(path) untuk membaca file menjadi data String
- Jangan membaca file ukuran besar dengan cara ini, karena akan menjadikan seluruh data di simpan di memory, yang bisa menyebabkan error OutOfMemory



## Kode : Membaca File Kecil

```
@Test
void readSmallFile() throws IOException {
    Path path1 = Path.of("small1.txt");
    byte[] bytes = Files.readAllBytes(path1);
    String content1 = new String(bytes);
    Assertions.assertEquals("Hello World", content1);

    Path path2 = Path.of("small2.txt");
    String content2 = Files.readString(path2);
    Assertions.assertEquals("Hello World", content2);
}
```

---

# Input Stream

---

# Input Stream

- InputStream merupakan base Interface yang digunakan untuk membaca Resource
- InputStream menggunakan numeric data, sehingga jika kita ingin membaca data text, kita harus konversi menjadi data numeric terlebih dahulu
- Sangat disarankan menggunakan InputStream pada kasus data yang non Text, seperti gambar, video, dan lain-lain
- Jika data Text, lebih disarankan menggunakan Reader karena bisa langsung menggunakan tipe data Character
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/InputStream.html>

---

# Membaca Data

- Untuk membaca data di InputStream, kita bisa gunakan method dengan awalan read()
- Pada kasus data sedikit, kita bisa membaca dengan menggunakan method read(), namun pada kasus data yang banyak, agar lebih cepat, kita bisa menggunakan method read(bytes) agar bisa membaca data sekaligus sebanyak panjang array bytes
- Jangan gunakan readAllBytes() pada kasus data besar, karena akan disimpan seluruh datanya di memory
- Method read() akan mengembalikan -1 jika sudah tidak ada data yang bisa dibaca lagi



## Kode : Input Stream Read

```
void inputStreamRead() {  
    Path path = Path.of("pom.xml");  
    try (InputStream stream = Files.newInputStream(path)) {  
        StringBuilder builder = new StringBuilder();  
        int data;  
        while ((data = stream.read()) != -1) {  
            builder.append((char) data);  
        }  
        System.out.printf(builder.toString());  
    } catch (IOException exception) {  
        Assertions.fail(exception);  
    }  
}
```



# Kode : Input Stream Read Bytes

```
void inputStream() {
    Path path = Path.of("pom.xml");
    try (InputStream stream = Files.newInputStream(path)) {
        StringBuilder builder = new StringBuilder();
        byte[] buffer = new byte[1024];
        int length;
        while ((length = stream.read(buffer)) != -1) {
            builder.append(new String(buffer, 0, length));
        }
        System.out.printf(builder.toString());
    } catch (IOException exception) {
        Assertions.fail(exception);
    }
}
```

---

# Output Stream

---

# Output Stream

- OutputStream merupakan base Interface yang digunakan untuk menulis ke Resource
- OutputStream menggunakan numeric data, sehingga jika kita ingin menyimpan data text, kita harus konversi menjadi data numeric terlebih dahulu
- Sangat disarankan menggunakan OutputStream pada kasus data yang non Text, seperti gambar, video, dan lain-lain
- Jika data Text, lebih disarankan menggunakan Writer karena bisa langsung menggunakan tipe data Character
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/OutputStream.html>

---

## Menulis Data

- Untuk menulis data di OutputStream, kita bisa gunakan method dengan awalan write()
- Kita bisa mengirim data dalam bentuk byte int, atau byte[] pada method write()

---

## Flush

- Secara default, pada saat kita melakukan close() OutputStream, data yang kita tulis akan di simpan secara permanen di target Resource
- Namun pada kasus kita banyak menulis data, sangat disarankan menggunakan method flush() untuk memaksa OutputStream untuk menyimpan data secara permanen ke target Resource
- Hal ini untuk waspada ketika terjadi system crash yang menyebabkan aplikasi berhenti, namun kita belum melakukan close() OutputStream, yang bisa menyebabkan data belum disimpan secara permanen ke Resource



# Kode : Output Stream

```
class Test {
    void outputStream() {
        Path path = Path.of("output.txt");
        try (OutputStream stream = Files.newOutputStream(path)) {
            for (int i = 0; i < 100; i++) {
                stream.write("Hello World\n".getBytes());
                stream.flush();
            }
        } catch (IOException exception) {
            Assertions.fail(exception);
        }
    }
}
```

---

# Reader



# Reader

- Reader adalah Input Stream untuk membaca resource berupa text
- Dengan menggunakan Reader, kita bisa melakukan konversi otomatis dari numeric data menjadi character data
- Untuk membaca data per character, kita bisa menggunakan method read()
- Sedangkan untuk membaca langsung beberapa character, kita bisa gunakan method read(char[])
- Cara penggunaannya mirip dengan InputStream
- [https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java.io/Reader.html](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Reader.html)



# Kode : Reader Read

```
void reader() {  
    Path path = Path.of("output.txt");  
    try (Reader reader = Files.newBufferedReader(path)) {  
        StringBuilder builder = new StringBuilder();  
        int data;  
        while ((data = reader.read()) != -1) {  
            builder.append((char) data);  
        }  
        System.out.println(builder.toString());  
    } catch (IOException exception) {  
        Assertions.fail(exception);  
    }  
}
```



# Kode : Reader Read Chars

```
void readerChars() {  
    Path path = Path.of("output.txt");  
    try (Reader reader = Files.newBufferedReader(path)) {  
        StringBuilder builder = new StringBuilder();  
        char[] buffer = new char[1024];  
        int length;  
        while ((length = reader.read(buffer)) != -1) {  
            builder.append(buffer, 0, length);  
        }  
        System.out.println(builder.toString());  
    } catch (IOException exception) {  
        Assertions.fail(exception);  
    }  
}
```

---

# Writer



# Writer

- Writer merupakan base Interface yang digunakan untuk menulis ke Resource untuk data text
- [https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java.io/Writer.html](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Writer.html)

---

## Menulis Data

- Untuk menulis data di Writer, kita bisa gunakan method dengan awalan write()
- Kita bisa mengirim data dalam bentuk char int, char[] atau String pada method write()



## Kode : Writer

```
@Test
void writer() {
    Path path = Path.of("writer.txt");
    try (Writer stream = Files.newBufferedWriter(path)) {
        for (int i = 0; i < 100; i++) {
            stream.write("Hello World\n");
            stream.flush();
        }
    } catch (IOException exception) {
        Assertions.fail(exception);
    }
}
```

---

# Open Option

---

# Open Option

- Saat kita membuat IO Stream menggunakan class Files, terdapat parameter diakhirnya yaitu OpenOption
- OpenOption merupakan opsi tambahan yang ingin kita lakukan pada proses IO Stream
- OpenOption merupakan sebuah interface, implementasinya kita bisa menggunakan enum StandardOpenOption
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/OpenOption.html>
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/file/StandardOpenOption.html>



# Kode : Open Option

```
@Test
void appendWithOpenOption() {
    Path path = Path.of("append.txt");
    try (Writer stream = Files.newBufferedWriter(path,
        StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {
        stream.write("Hello World\n");
        stream.flush();
    } catch (IOException exception) {
        Assertions.fail(exception);
    }
}
```

---

# Object Stream

---

# Java Bean

- Saat kita membuat program Java, kita sering membuat data dalam bentuk Java Bean (class dengan getter dan setter)
- Pada kasus kita ingin menyimpan data object tersebut ke file, jika kita menggunakan Input/Output Stream, maka akan sangat menyulitkan, karena kita harus lakukan secara manual, dan melakukan konversi secara manual
- Untungnya Java memiliki fitur Object Stream, dimana kita bisa menyimpan/membaca data Java Object dari file

---

# Serializable

- Salah satu syarat ketika ingin membuat Java Bean Class yang bisa disimpan kedalam file hasil objectnya, kita harus membuat Java Bean Class turunan dari interface Serializable
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Serializable.html>
- Setelah itu, kita harus memberi tahu versi dari class tersebut dengan menggunakan static attribute static final long serialVersionUID, dimana berisikan versi dari Java Class nya
- Setiap kita melakukan perubahan di class nya, misal menambahkan attribute, menghapus attribute, maka kita harus mengubah versi nya, agar tidak terjadi masalah ketika melakukan Object Stream



## Kode : Person Class

```
no usages new *
6 public class Person implements Serializable {
    no usages
    7     public static final long serialVersionUID = 1L;
8
9     2 usages
10    private String id;
11
12    2 usages
13    private String name;
```



# Object Stream

- Untuk menyimpan data object, kita bisa menggunakan ObjectInputStream
- [https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java.io/ObjectInputStream.html](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/ObjectInputStream.html)
- Dan untuk mengambil data object, kita bisa menggunakan ObjectOutputStream
- [https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java.io/ObjectOutputStream.html](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/ObjectOutputStream.html)



# Kode : Object Output Stream

```
@Test
void savePerson() {
    Person person = new Person();
    person.setId("1");
    person.setName("Eko");
    Path path = Path.of("eko.person");

    try (OutputStream stream = Files.newOutputStream(path);
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(stream)) {
        objectOutputStream.writeObject(person);
        objectOutputStream.flush();
    } catch (IOException exception) {
        Assertions.fail(exception);
    }
}
```



# Kode : Object Input Stream

```
@Test
void getPerson() {
    Path path = Path.of("eko.person");
    try (InputStream stream = Files.newInputStream(path)) {
        ObjectInputStream objectInputStream = new ObjectInputStream(stream));
        Person person = (Person) objectInputStream.readObject();
        Assertions.assertEquals("1", person.getId());
        Assertions.assertEquals("Eko", person.getName());
    } catch (IOException | ClassNotFoundException exception) {
        Assertions.fail(exception);
    }
}
```

---

## Java Collection

- Hampir semua class Java Collection seperti List, Set, Map merupakan turunan dari Serializable
- Artinya kita bisa menyimpan data collection tersebut di file menggunakan Object Stream
- Dengan syarat isi data nya adalah data Serializable juga

---

# Memory Stream

---

## Memory Stream

- Kadang ada kasus kita ingin membuat IO Stream, namun target resource nya tidak di file, misal hanya di variable di memory
- Pada kasus seperti itu, kita bisa memanfaatkan Memory Stream yang tersedia di Java

---

## Contoh Memory Stream

- ByteArrayInputStream, turunan InputStream yang mengambil datanya dari byte[]
- ByteArrayOutputStream, turunan OutputStream yang menyimpan datanya di byte[] di memory
- StringReader, turunan Reader yang mengambil data dari String
- StringWriter, turunan Writer yang menyimpan data di String di memory



## Kode : Memory Stream

```
@Test
void memoryStream() {
    StringWriter writer = new StringWriter();
    for (int i = 0; i < 10; i++) {
        writer.write("Hello World\n");
    }

    String content = writer.getBuffer().toString();
    System.out.println(content);
}
```

---

# Print Stream



# Print Stream

- Salah satu turunan dari Output Stream yang sering kita gunakan secara tidak sadar adalah Print Stream
- Print Stream adalah Output Stream yang bisa menerima berbagai jenis data, dan secara otomatis melakukan konversi ke numeric data yang dibutuhkan oleh Output Stream
- Saat kita menggunakan System.out, itu sebenarnya adalah object dari Print Stream
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/PrintStream.html>

---

## Kode : Print Stream Console

```
class PrintStream {
    void printStream() {
        PrintStream stream = System.out;

        stream.println("Hello World");
        stream.println("Hello World");
        stream.println("Hello World");
    }
}
```



## Kode : Print Stream File

```
@Test
void printStreamFile() {
    Path path = Path.of("print.txt");
    try (OutputStream outputStream = Files.newOutputStream(path)) {
        PrintStream stream = new PrintStream(outputStream) {
            stream.println("Hello World");
            stream.println("Hello World");
            stream.println("Hello World");
        } catch (IOException exception) {
            Assertions.fail(exception);
        }
    }
}
```

---

# Scanner



# Scanner

- Scanner adalah class mirip seperti PrintStream, namun khusus untuk membaca dari InputStream
- Dengan menggunakan Scanner, kita bisa membaca dan melakukan konversi tipe data secara otomatis
- Method dengan awalan next() digunakan untuk membaca data
- Method dengan awalan has() digunakan untuk mengecek apakah masih ada data
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html>



## Kode : Scanner

```
void scannerFile() {  
    Path path = Path.of("print.txt");  
    try (InputStream stream = Files.newInputStream(path);  
        Scanner scanner = new Scanner(stream)) {  
        while (scanner.hasNextLine()) {  
            String line = scanner.nextLine();  
            System.out.println(line);  
        }  
    } catch (IOException exception) {  
        Assertions.fail(exception);  
    }  
}
```

---

## Scanner Console

- Scanner juga banyak digunakan untuk membaca input dari console
- Dimana System.in merupakan tipe data InputStream, yang bisa kita gunakan sebagai input untuk Scanner

---

## Kode : Scanner Console

```
▶ public class InputApp {  
    no usages new *  
▶     public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Masukkan nama: ");  
        String name = scanner.nextLine();  
        System.out.println("Halo " + name);  
    }  
}
```

---

# IO Stream Lainnya

---

## IO Stream Lainnya

- Sebenarnya masih banyak class IO Stream lainnya yang belum dibahas di kelas ini
- Namun dasar-dasarnya sama, InputStream dan OutputStream untuk numeric data, dan Reader dan Writer untuk character data
- Masih banyak class turunannya, namun kebanyakan digunakan untuk mempermudah kita ketika menggunakan IO Stream, mirip seperti PrintStream atau Scanner
- Untuk lebih detail, silahkan eksplorasi package java.io
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/package-summary.html>