

# Java Standard Class

**Silver** - Chapter 2 - Topic 6

Selamat datang di **Chapter 2 Topic 6**  
online course **Back End Java** dari  
Binar Academy!



### Wow, you've reached the last topic!

Selamat! karena kamu udah menjelajahi chapter 2 sampai di topik terakhir 🎉

Untuk melengkapi pemahaman kita, pada topik ini kita bakal cari tahu tentang **Java Standard Class**.

Kalau gitu, gimana kalau kita langsung geser aja slide-nya?



**Dari sesi ini, harapannya kamu bisa mendapatkan beberapa hal berikut!**

- String
- Date and Calendar class, Timezone, localDate
- Math Class
- Regular expression
- Write and read .txt dan .csv file



Okay, First thing first, kita bakal belajar dulu tentang String.

Penjelasannya bakal dibagi ke beberapa bagian nih, ada **String**, **StringBuffer**, **StringBuilder**, dan **StringJoiner**.

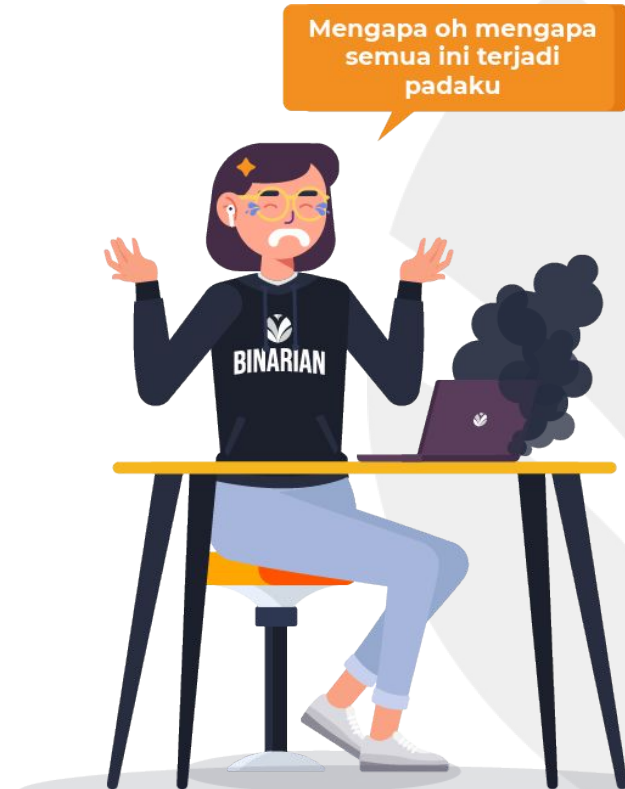


### Kamu masih ingat kalau kita pernah bahas tentang tanda tambah “+” untuk menggabungkan string?

Yap, tanda tambah atau “+” memang bisa untuk menggabungkan string.

Sayangnya, kalau kita mau menggabungkan string yang panjang, cara ini nggak bakal cocok buat dipakai.

Iya, karena **penggabungan string memakan memory dan bakal memperlambat proses.**



Kita pakai contoh, ya. Misalnya ada dua object yang berbeda, yaitu “Makan” dan “Nasi”.

**String result = “Makan”+”Nasi”;**

Terus kalau kita mau menggabungkan kedua object ini, berarti nantinya **bakal jadi object baru** berupa “MakanNasi”



Oh iya, selain pakai "+", sebenarnya kita juga bisa pakai **concat()**

```
String result = "Makan".concat("Nasi");
```





Dari contoh tadi, kita udah menciptakan 3 object yang berbeda nih, yaitu:

- Makan
- Nasi
- MakanNasi



Kalau kata yang dipakai masih sedikit, mungkin bikinnya masih under control yaaa~

Tapi bakal jadi masalah kalau kita mau menggabungkan beribu-ribu kata. Bakalan banyak banget object yang bakal kita buat?! Cape deh~



Kabar baiknya, proses penggabungan String atau **concatenation** punya berbagai cara di Java.

That's why, kita nggak cuma bisa pakai tanda “+” atau `concat()` aja.

Melainkan, melalui beberapa class yang udah disediakan Java untuk memodifikasi String secara best practice.



### Emangnya concatenation ini penting, ya?

Yes! String merupakan sebuah object yang bersifat immutable.

Eitsss, bukan imut yang artinya lucu-lucu gitu, ya!

**Immutable ini artinya object String yang udah dibuat nggak bakal bisa dimodifikasi.** Jadi, mau nggak mau kamu harus bikin object yang baru lagi karena udah nggak bisa diubah.



**Karena nggak bisa dimodifikasi, jadi  
Java bikin class-class biar  
concatenation jadi lebih efektif~**

Mulai ketangkep benang merahnya, ya.

Java menyediakan **StringBuffer** dan **StringBuilder** sebagai solusi dari **concatenation** dalam jumlah yang besar.



Tapi perlu diingat juga, dengan dipakai dalam jumlah yang besar, berarti penggunaannya jadi nggak efektif kalau kamu mau gabungin object yang sedikit.



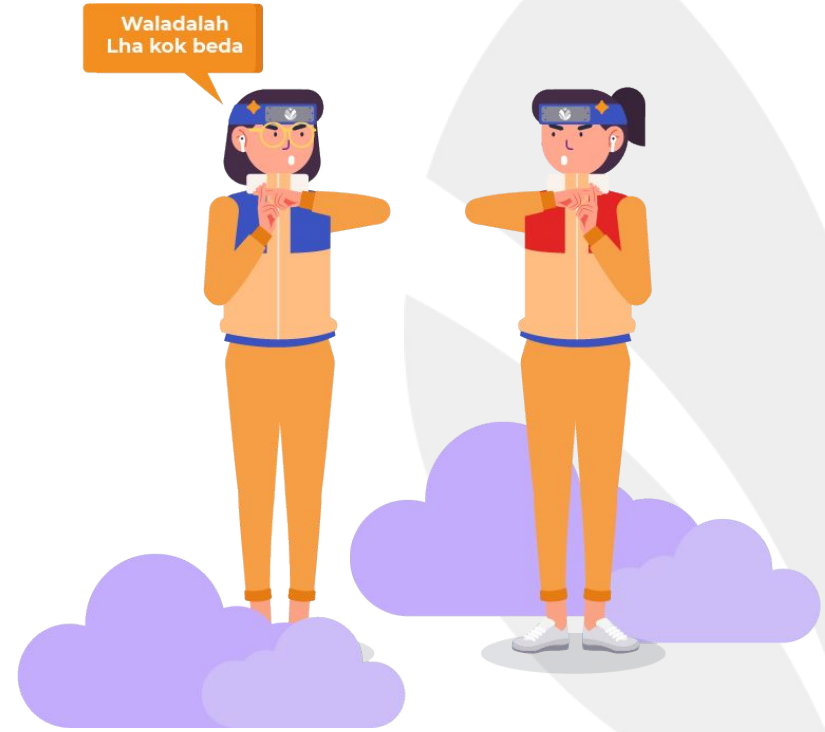
### Meskipun mirip, tapi **StringBuffer** dan **StringBuilder** punya beberapa perbedaan, lho~

- **StringBuffer** merupakan object yang thread safe, sedangkan **StringBuilder** tidak thread safe.

**Thread safe** adalah aman dipakai di program yang pakai multithreading.

Penjelasan tentang multithreading bakal kita bahas di chapter 7, ya.

- **StringBuilder** punya **performance yang lebih baik** dibandingkan dengan **StringBuffer**.




Kalau pakai kedua class ini, pas ada penggabungan string baru, nggak perlu ada object baru yang dibuat. Iya, karena semuanya disatukan jadi satu object aja.

Canggih, kan?






Berikut adalah contoh penggunaan StringBuffer:



```
StringBuffer sb = new StringBuffer("");  
sb.append("Java ");  
sb.append("Backend");  
String str = sb.toString();  
System.out.println("String object: "+str);
```

Kalau yang ini adalah contoh dari penggunaan StringBuilder:



```
StringBuilder sb = new StringBuilder("");  
sb.append("Java ");  
sb.append("Backend");  
String str = sb.toString();  
System.out.println("String object: "+str);
```

### Selain ada StringBuffer dan StringBuilder, ada juga yang namanya StringJoiner

StringJoiner adalah Class String di Java 8 yang memungkinkan kamu untuk **menggabungkan string pakai delimiter**.

Dengan class ini kita bisa gampang banget buat bikin csv file.



Penasaran sama contohnya?

Berikut adalah contoh penggunaan StringJoiner:

Inputnya

```
List<String> rgbList = new ArrayList<>();
rgbList.add("Red");
rgbList.add("Green");
rgbList.add("Blue");
StringJoiner rgbJoiner = new
StringJoiner(",");
for (String color : rgbList) {
    rgbJoiner.add(color);
}
System.out.println(rgbJoiner.toString());
```

Outputnya

Red,Green,Blue

Pada constructor dari `StringJoiner`, kita bisa lho menambahkan delimiter, prefix dan suffix.

Contohnya kayak gini, nih:



```
StringJoiner rgbJoiner = new  
StringJoiner(", ", PREFIX, SUFFIX);
```

Thank you String for your existence~

Lanjut nih, kita bakal bahas tentang manipulasi.

Serem, ya? Manipulasinya sih ada kaitannya sama **Date**, **Calendar Class**, **Timezone**, dan **localDate**. Yuk kita investigasi 🤔

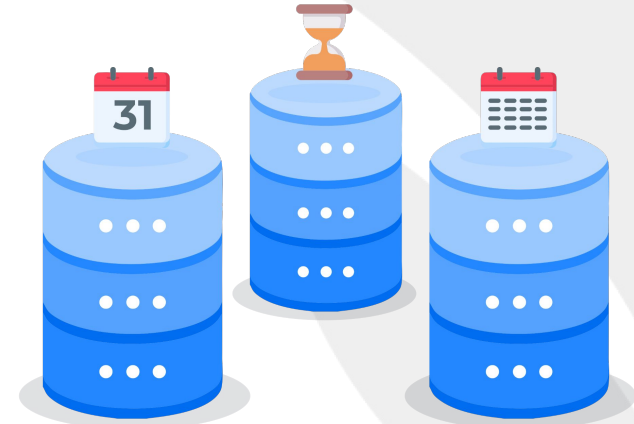


### Bukannya berisi buku, library pada Java bisa memanipulasi data juga, lho!

Untuk memanipulasi data berupa waktu, tanggal, dan timestamp, si Java ini punya library yang bernama **Date, Calendar dan Time**.

Dengan library ini, kita bisa mengatur secara detail waktu, zona waktu, dan juga formatnya.

Penggunaan library ini secara detail bisa kamu cek di website berikut: [Java Date Time Tutorials](#)



Next, siapa disini yang suka belajar matematika?

Walaupun love-hate sama matematika, tapi pelajaran ini udah ngebantu kita banyak banget!

Salah satunya di Java. Ada **Math Class** yang membuat kita kembali belajar tentang operasi matematika.





Di sini Java bakal bikin matematika yang dirasa sulit bakal jadi gampang banget.

Serius! Soalnya untuk melakukan operasi matematika, Java udah menyediakan class yang namanya **Math**.

**Berfungsi untuk mempermudah beberapa operasi yang cukup rumit.**

Nah, kalau kamu tertarik untuk pakai class Math secara lengkap, kamu bisa akses di website ini: [Java Math Class](#)



Math Class won't be same anymore~

Eh, tapi kalian kerasa nggak sih kalau kita udah hampir selesai di topic ini?

Iya, dikit lagi. Setelah ini ada **Regular Expression** yang menunggu kamu.



**Regular Expression** atau Regex adalah sebuah String yang berisi suatu pola atau karakter untuk melakukan pencocokkan.

Regex biasanya dipakai untuk melakukan **validasi** atau pencarian.



Kita nggak perlu hafal tentang simbol dari Regex, tapi kalau kita tahu basic dari regex, itu bakal bantu kita buat melakukan validasi ataupun pencocokkan.

Penggunaan Regex diatur Java di **library java.util**

Berikut penjelasan secara lengkap mengenai penggunaan [Java Regex](#)



Dari Regex kita move on ke format file.

Mirip-mirip kayak .pdf dan .doc, kali ini kita bakal jelasin format file .txt dan .csv.

Oh iya, format diatas sekaligus jadi bagian dari materi terakhir kita, yaitu **Write and Read File (.txt dan .csv)**.



Chingu, pada subtopic ini kita bakal belajar caranya membaca suatu file pakai library yang udah disediakan oleh Java.

File yang dibaca dalam pembahasan kali ini cuma terbatas pada extension **csv** dan **txt** aja.



### Kamu sering pakai aplikasi pengolah data kayak Microsoft Excel atau Google Spreadsheet?

Kalau iya, kamu pasti udah nggak asing lagi sama file csv. File csv punya nama panjang berupa **Comma Separated Values**.

Sesuai dengan namanya, file csv ini adalah **file yang teratur** karena punya kolom yang dibatasi sama **delimiter**.



**Delimiter adalah pembatas antara data yang satu dengan data lainnya.** Salah satu contoh dari delimiter adalah koma (,).

Ibaratnya kalau kamu mau menyebutkan nama hewan yang lebih dari satu, berarti kamu bisa pakai koma untuk memisahkan nama hewan antara satu dengan yang lainnya.

Misalnya, gajah, kucing, ikan, dan kuda.





Untuk menulis sebuah file csv, kita dapat memanfaatkan berbagai macam perulangan dengan bantuan object seperti list.

Satu line csv tersebut bisa dianggap sebuah object dan kolom-kolomnya bisa dianggap sebagai fieldnya.

Object tersebut bakal disimpan di dalam sebuah list yang nantinya bakal jadi baris-baris pada csv.



### Kita pakai contoh ya!

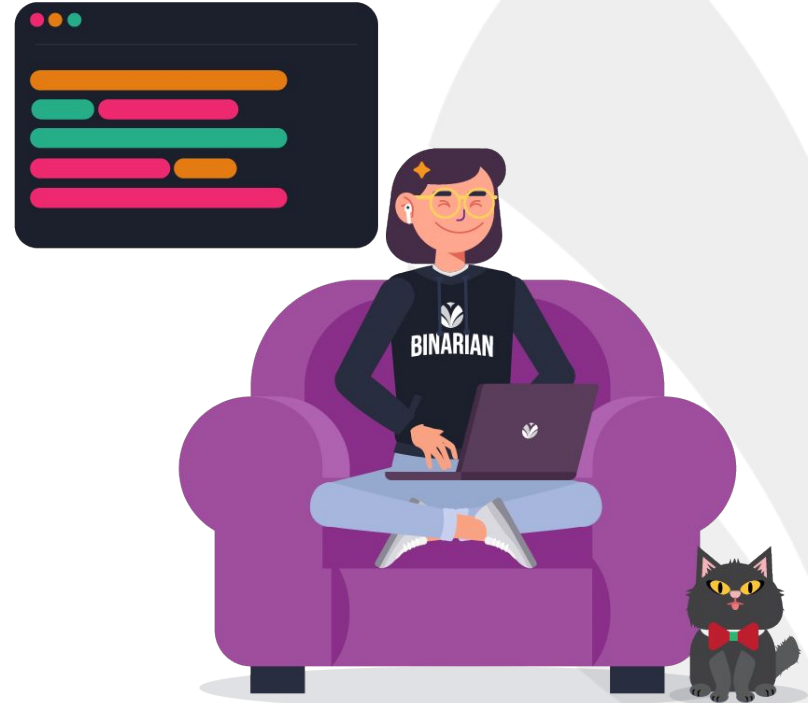
Misalnya ada kolom berikut:

Nama, Tanggal Lahir, Alamat rumah

Data dari kolom tersebut bakal kita petakan jadi field Nama, Tanggal Lahir dan Alamat. Semuanya bakal kita simpan dalam bentuk String di sebuah object, misalnya **CsvRecord**.

Nantinya Object tersebut diletakkan di sebuah list yang jadi **List<CsvRecord>**

Object inilah yang bakal melakukan iterasi buat penulisan di csv~



### Kamu tahu nggak, kalau ternyata Java udah menyediakan library sendiri?

Iya, serius! Namanya **java.io** yang tugasnya adalah mengelola input dan output di Java kayak gini:

- Untuk membaca file atau menampilkan output, kita pakai **BufferedReader**.
- Sedangkan untuk menulis file atau memasukkan input, kita pakai **BufferedWriter**.



Coba kamu perhatikan method read yang dipakai untuk baca csv file pada gambar di bawah, ya!

```
public static void read(String csvFile) {  
    try {  
        File file = new File(csvFile);  
        FileReader fr = new FileReader(file);  
        BufferedReader br = new BufferedReader(fr);  
        String line = "";  
        String[] tempArr;  
        while((line = br.readLine()) != null) {  
            tempArr = line.split(delimiter);  
            for(String tempStr : tempArr) {  
                System.out.print(tempStr + " ");  
            }  
            System.out.println();  
        }  
        br.close();  
    } catch(IOException ioe) {  
        ioe.printStackTrace();  
    }  
}
```

Dari gambar tersebut, kita bisa menemukan beberapa hal kayak gini:

- Parameter dari method tersebut merupakan lokasi dari file atau absolute path dari file csv-nya dalam bentuk data String.
- **Absolute path** merupakan path dari suatu file yang lengkap banget. Contohnya kayak gini, C:/Temp/binar.csv

```
public static void read(String csvFile) {  
    try {  
        File file = new File(csvFile);  
        FileReader fr = new FileReader(file);  
        BufferedReader br = new BufferedReader(fr);  
        String line = "";  
        String[] tempArr;  
        while((line = br.readLine()) != null) {  
            tempArr = line.split(delimiter);  
            for(String tempStr : tempArr) {  
                System.out.print(tempStr + " ");  
            }  
            System.out.println();  
        }  
        br.close();  
    } catch(IOException ioe) {  
        ioe.printStackTrace();  
    }  
}
```

### Terus, cara baca file-nya gimana?

Tenang, tenang~

Jadi, untuk membaca file-nya kayak gini, nih:

1. Object file bakal baca absolute path tersebut dan di convert jadi bentuk path yang bisa diolah sama Java.
2. Kemudian FileReader bakal mendeteksi file pake parameter object dari file.



3. BufferedReader bakal mendeteksi line per line pakai method `readLine()` dan bakal mendeteksi line per line sebagai suatu String.
4. Lalu, String per line tersebut dijadikan sebuah array yang bakal di print out di console yang udah dipisahkan dengan spasi.

Hal ini bakal berlanjut selama BufferedReader masih menemukan line. Kalau BufferedReader udah nggak menemukan line, berarti hasilnya bakal null.

3. BufferedReader bakal memanggil method `close()` buat mengakhiri pembacaan file csv.

```
public static void read(String csvFile) {  
    try {  
        File file = new File(csvFile);  
        FileReader fr = new FileReader(file);  
        BufferedReader br = new BufferedReader(fr);  
        String line = "";  
        String[] tempArr;  
        while((line = br.readLine()) != null) {  
            tempArr = line.split(delimiter);  
            for(String tempStr : tempArr) {  
                System.out.print(tempStr + " ");  
            }  
            System.out.println();  
        }  
        br.close();  
    } catch(IOException ioe) {  
        ioe.printStackTrace();  
    }  
}
```

Seluruh method diletakkan di **try-catch block**, supaya jika ada kegagalan ketika pembacaan file, nggak ada error program yang terjadi.

Tapi, walaupun memang terjadi error di proses input atau output-nya, bakal ada sebuah exception dari **IOException**. Jadi nggak usah panik duluan yaaa~

Pembahasan lebih lengkap tentang Try Catch block dan Exception ini bakal dibahas di chapter berikutnya.





Kalau tadi udah bahas buat file csv. Sekarang, kita coba perhatikan method write yang dipakai untuk membaca txt file, ya!

- Parameter method tersebut bakal jadi absolute path yang bakal dipakai untuk menaruh file dan memberikan nama file.

Contohnya: C:/Temp/binar.txt.

- Kalo kamu mau bikin file csv, tinggal ganti aja extension filenya jadi csv. Jadi, kamu nggak perlu bikin ulang deh.

```
public static void write(String txtFile) {
    try {
        File file = new File(txtFile);
        if (file.createNewFile()){
            System.out.println("new file is created");
        }
        FileWriter writer = new FileWriter(file);
        BufferedWriter bwr = new
        BufferedWriter(writer);
        bwr.write("Binar");
        bwr.newLine();
        bwr.flush();
        bwr.close();
        System.out.println("succesfully written to a
        file");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

- Ada lagi, nih. Object File dibuat pakai parameter constructor absolute path. Buat bikin file baru, kita bisa pake method `createNewFile()`.

Kalau file yang namanya udah ada di direktori tersebut, maka `createNewFile` bakal mengembalikan `false`.

```
public static void write(String txtFile) {
    try {
        File file = new File(txtFile);
        if (file.createNewFile()){
            System.out.println("new file is created");
        }
        FileWriter writer = new FileWriter(file);
        BufferedWriter bwr = new
        BufferedWriter(writer);
        bwr.write("Binar");
        bwr.newLine();
        bwr.flush();
        bwr.close();
        System.out.println("succesfully written to a
        file");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

Kayak `FileReader`, kita butuh `FileWriter` untuk menulis file tersebut. Hal ini nantinya sekaligus jadi parameter `BufferedWriter`.

`Bufferwriter` bakal menulis karakter pakai method `write` dengan parameter `String`.

Terus kalau mau mulai line baru, kita bisa pakai method `newLine`.

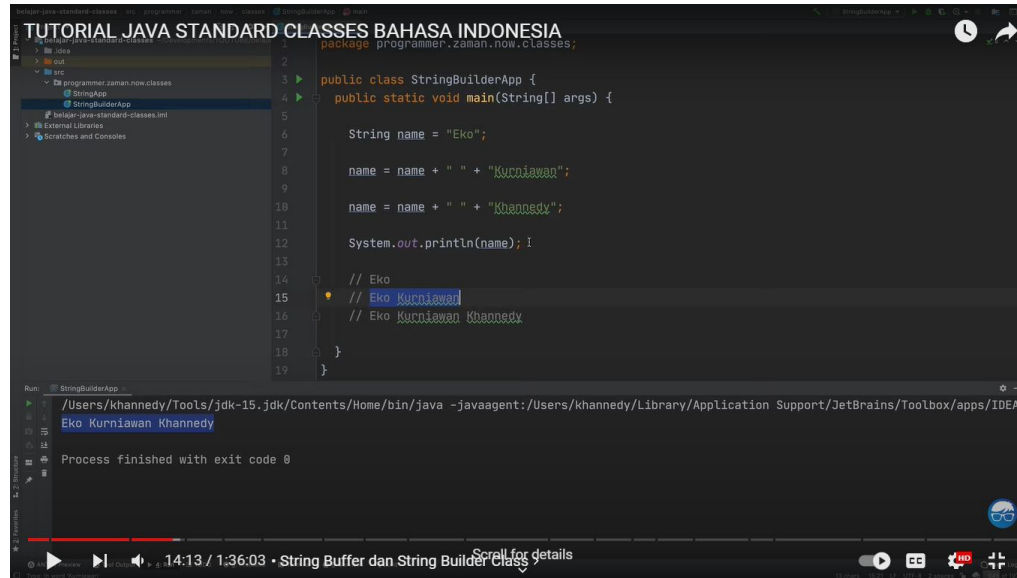
```
public static void write(String txtFile) {
    try {
        File file = new File(txtFile);
        if (file.createNewFile()){
            System.out.println("new file is created");
        }
        FileWriter writer = new FileWriter(file);
        BufferedWriter bwr = new
        BufferedWriter(writer);
        bwr.write("Binar");
        bwr.newLine();
        bwr.flush();
        bwr.close();
        System.out.println("succesfully written to a
        file");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

- Lanjut~ setiap kali file mau ditutup dan selesai menulis, berarti method `flush()` lebih baik dijalankan.

Hal itu diperlukan supaya apa yang udah ditulis pake object `BufferedWriter` ini nggak mengganggu pas menulis file lain yang object-nya sama.

- Untuk menyelesaikan penulisan file, kita harus pakai method `close()`;
- Sama kayak pas melakukan pembacaan file, sebuah input, output code block-nya harus diletakkan di dalam try-catch, ya!

```
public static void write(String txtFile) {
    try {
        File file = new File(txtFile);
        if (file.createNewFile()){
            System.out.println("new file is created");
        }
        FileWriter writer = new FileWriter(file);
        BufferedWriter bwr = new
        BufferedWriter(writer);
        bwr.write("Binar");
        bwr.newLine();
        bwr.flush();
        bwr.close();
        System.out.println("succesfully written to a
        file");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```



```
package programmer.zaman.now.classes;

public class StringBuilderApp {
    public static void main(String[] args) {

        String name = "Eko";

        name = name + " " + "Kurniawan";

        name = name + " " + "Khannedy";

        System.out.println(name);
    }

    // Eko
    // Eko Kurniawan
    // Eko Kurniawan Khannedy
}
```

Run: /Users/khannedy/Tools/jdk-15.jdk/Contents/Home/bin/java -javaagent:/Users/khannedy/Library/Application Support/JetBrains/Toolbox/apps/IDEA-...  
Eko Kurniawan Khannedy  
Process finished with exit code 0

Sebagai penutup topik ini, berikut ada video tutorial Java Standard Class yang bisa kamu tonton. Mulai dari String Class, Math Class, Date and Calendar Class, kamu bisa pakai video ini sebagai referensi. Jangan lupa untuk mengalokasikan waktu untuk menonton videonya, ya~

[Tutorial Java Standard Class](#)

Selamat karena kamu sudah berjuang sampai di akhir topic Chapter 2 🙌

Dari seluruh materi yang sudah kamu pelajari di Chapter 2, materi mana yang menurutmu paling menarik? Lalu, alasannya kenapa?



Congratulations! Kita udah sampai di penghujung chapter~

Pada chapter selanjutnya, kita bakal belajar tentang Error dalam **Java Unit Test**.

Siapa yang udah nggak sabar move on ke chapter 3? See you soon 🏃 🏃

