



Java: String dan Array

IF2210 – Semester II 2024/2025

by: RSP & IL; rev: SAR

String

String

- › String merupakan kelas khusus di Java.
 - › Java memperlakukan String tidak seperti objek lain.
 - › Bukan tipe dasar, tapi bagian dari bahasa Java.
- › String dibahas karena akan banyak dipakai (misalnya untuk menuliskan output).
 - › `String s = "Hello World";`
- › String merupakan sebuah kelas, dan di dalamnya ada beberapa *method*, misalnya:
 - › `s.length()` → panjang string.
 - › `s.substr(0, 2)` → mengembalikan "He" (karakter ke-0 sampai sebelum 2 [bukan sampai dengan 2]).

Operator String

- › Selain memiliki method, string juga memiliki operator “+”.
- › Operator + akan mengkonversi float/integer secara otomatis jika digabung dengan string.

```
/*konversi otomatis 2 ke string*/
String s = "Banyaknya "+ 2;

int z = 4;
String s = "Ada " + z + " buah ";

String s = 5; /*tidak boleh */
```

Membandingkan String

- › Method `equals()` membandingkan `String` untuk memeriksa kesamaan.
- › Method `equalsIgnoreCase()` melakukan hal yang sama, tapi besar kecil huruf tidak diperhatikan.
- › Method `compareTo()` menghasilkan perbandingan urutan `String` menurut kamus (*lexicographically*):
 - › 0 jika string sama,
 - › >0 jika `String1>String2`, dan
 - › <0 jika `String1<String2`

Konversi String ke bilangan

```
int x = Integer.parseInt("2");
long l = Long.parseLong("2L");
double d = Double.parseDouble("2.0");
float f = Float.parseFloat("2.0f");
```

```
String ival = Integer.toString(2);
String lval = Long.toString(2L);
String dval = Double.toString(2.0);
String fval = Float.toString(2.0f);
```

Literal String

- › Literal String merupakan instans dari objek **String**.
- › Method boleh dipanggil langsung dari Literal:
 - › "Hello".length() menghasilkan 5.

Sekuens escape String

- › Serangkaian karakter diawali \ (backslash) untuk mengetikkan karakter khusus, escape berikut sama dengan C/C++.
 - › \n : newline
 - › \t : karakter tab
 - › \\ : backslash
 - › \" : double quote
 - › \' : apostrophe
- › Escape yang tidak ada di C/C++: \udddd: karakter dalam unicode. dddd adalah digit heksadesimal (0-9, A-F). Contoh:
"h\u00e9jo" sama dengan "héjo"

Sifat Immutable String

- › String sebenarnya immutable (tidak bisa diubah).
- › Dalam instruksi sbb:

```
String a = "hello";
a = a + " world";
```

- › Sebuah objek baru diciptakan, objek lama dibuang (untuk dipungut oleh *garbage collector*). a menunjuk ke objek yang baru.

Operasi String Tidak Optimal

- › String baru diciptakan (String yang lama tetap ada di memori, dan dibuang ketika terjadi *garbage collection*).
- › Jika semua variabel di bawah adalah String, maka ekspresi ini menciptakan dan membuang beberapa objek antara:

```
a = a + b + c + d;
```
- › Untuk operasi yang banyak melibatkan perubahan string, sebaiknya menggunakan **StringBuilder**.

StringBuilder

- › **StringBuilder** adalah penggunaan *design pattern* “builder” untuk membantu penciptaan **String**.
- › Sifatnya *mutable*.
- › Tidak ditangani secara transparan oleh Java (harus dilakukan secara manual).
 - › Tidak bisa `StringBuilder sb = "hello";`
- › Lebih cepat untuk manipulasi **String** yang memerlukan perubahan pada **String**.

Sifat mutable StringBuilder

- › Untuk mengubah `StringBuilder` tidak perlu objek baru.
 - › Contoh :

```
StringBuilder sb = new StringBuilder("mataku");
sb.setCharAt(4, 'm');
sb.append(" biru");
String s = sb.toString(); // "matamu biru"
```

Untuk mengubah `String` selalu butuh objek baru (objek lama diubah melalui *assignment*).

Sekilas: fluent interface, method chaining

- › Dalam *pattern “builder”* seringkali digunakan *fluent interface*, di mana method me-return `this`.
- › Hal ini memungkinkan *method chaining*, di mana kode ini:

```
StringBuilder sb = new StringBuilder("mataku");
sb.replace(4, 5, 'm');
sb.append(" biru");
```

dapat juga ditulis sebagai:

```
StringBuilder sb = new StringBuilder("mataku")
    .replace(4, 5, 'm')
    .append(" biru");
```

Method yang penting

- › Beberapa method **String** dan **StringBuilder** yang penting adalah:
 - › `length()`: panjang string yang sedang “dipegang” oleh builder
 - › `replace()`: mengganti suatu karakter
 - › `charAt()`: mengakses karakter di posisi tertentu
 - › `trim()`: menghilangkan spasi di awal dan di akhir string
- › Perhatikan bahwa meskipun nama method-nya sama, sifat keduanya berbeda.
 - › **String** menciptakan objek baru, sedangkan **StringBuilder** tidak.

Bacaan tentang Mutable dan Immutable Object

<http://www.javaranch.com/journal/2003/04/immutable.htm>

Method String `toString()`

- › Kelas `Object` memiliki method `toString()`.
- › *Override* method `toString()` untuk mencetak objek dengan lebih baik.
- › Misal, untuk kelas `Point`, isi method-nya:

```
String toString() {  
    return String.format("(%d,%d)", this.x, this.y);  
}
```

- › Dengan method di atas, `Point` bisa dicetak dengan mudah:

```
Point p = new Point(1,2);  
System.out.println(p); /*mencetak point*/
```

- › output potongan kode di atas: (1, 2).

Array

Array C++

- › Array sebagai type “primitif”, bukan Kelas, dalam bahasa C++ sama dengan Bahasa C.
- › Deklarasi :
 - › Dinamik dengan *pointer*: `int * T;`
 - › Statik, ukuran sudah ditentukan : `int X[10];`
- › Lihat kuliah slides Bahasa C (struktur data) untuk *array* dalam C++.
- › Baca tutorial program kecil Array C++.
- › Tidak disarankan untuk memakai *array* di “luar” kelas.

Array di Java: array adalah objek

- › Di Java, *array* adalah Objek.
- › Contoh *array*:

```
char ac[] = { 'n', 'o', 't', ' ', 'a', ' ', 's', 't',  
'r', 'i', 'n', 'g' };
```

- › *member* yang ada: `length` dan semua *method* yang ada di kelas `Object`.
- › dalam *array* di atas:

```
ac.length == 12
```

Array dinamis

- › Array diciptakan dengan new.

- › Contoh *array of integer*:

```
int a[] = new int[5];
```

- › Array tidak bisa di-resize. Untuk mengubah ukuran array:

1. buat *array* baru,

2. salin isi *array* lama ke yang baru.

Array of Objects

- › new pada array hanya mengalokasikan array, kode:

```
Lingkaran ling = new Lingkaran[5];
```

- › Hanya menciptakan *array of lingkaran* (lingkarannya sendiri belum diciptakan, perhatikan bahwa ini berbeda dengan C++).

- › To be exact: hanya mengalokasikan 5 *reference* ke objek bertipe Lingkaran, belum mengalokasikan objek-objek bertipe Lingkaran.

- › Untuk membuat 5 Objek Lingkaran dalam *array ling*:

```
for(int i=0; i<5; i++)
    ling[i] = new Lingkaran();
```

Array Multidimensi

- › Array multidimensi diakses dengan operator [] sebanyak dimensinya.
- › Contoh:

```
int matriks[][] = new int[5][6];
```

- › Untuk *array* yang ukuran tiap barisnya tidak sama:

```
int matrix[][] = new int[5][];  
matrix[0] = new int[5];  
matrix[1] = new int[3];
```

Array of Object

- › Perhatikan bahwa :
 - › *Array* harus dialokasi.
 - › Objek harus dialokasi.
- › Sebagai latihan:
 - › Definisikan kelas Point.
 - › Buatlah *array of Point* berkapasitas 10.
 - › Inisialisasi dengan Point of Origin.