Strings

Ch. 6

in Java, We treat strings as an object not as an Array.

Way 1

```
String name = new String("Ahmad");
```

or

```
String name = "Ahmad";
```

Way 2

```
char[] arr = {'A', 'h', 'm', 'a', 'd'};
String name = new String(arr);
```

Note Strings are Immutable. (لا نغير المخزون، لكن عكن ان أعيق البوينتر ليؤشر على شيء آخر)

ex
```
String one = "Ahmad";
String two = "Ahmad";
String three = new String("Ahmad");
...println(one.equals(two));
...println(one == two);
```
→ true
true ← يفحص اذا يشيران الى المكان نفسه

يفحص المحتوى

```
...println(one.equals(three));
...println(one == three);
```
→ true
false

Rule
```
String y = "Hi";
String x = "Hi"
```
~~String x = "Hi"~~ ← اذا كان لهما نفس القيمة

الـ String الذي ننشئه بالطريقة السريعة ينشئ واحدة، ولجعل البقية يؤشر عليها.
لكن الذي ننشئه بطريقة الـ object لا يؤشر عليها.

---

ex
```
String x = new String("Hi");
String y = new String("Hi");
...println(x.equals(y));
...println(x == y);
```
output

→ true
false

حتى لو انها علاقة من النوع نفسه ، object
لكن خاصية تعداد المؤشر هي فقط ←
للطريقة السريعة.

Rule You can't edit strings; it is Immutable

ex

```
String one = "Hi";
String two = "Hi";
String three = " bye";
two = three;

__println(one);
__println(two);
__println(three);
```

output →

Hi
bye
bye

(*) لا يمكن التعديل على string ، لذلك عندما غيرنا مؤشر two ، رغم ان كلمة Hi ، يؤشر عليها one,two لكن لم تتحول عندما غيرنا على two.

(*) ذلك بسبب ان التعديل على two هو تغير للتأشير من على Hi الى three يؤشر معها على bye

(*) لا يتأثر الكلمة الاصلية ولا مؤشر one .

(*) التغير في مؤشر two لا يتنافى مع Immutable

(*) يتغير التأشير، ولا تتغير القيمة الاصلية المخزنة

ex

```
String x = " Hi";
x = " bye";
```

} error ; Immutable

String y = "hc";

ex String x = "hi";

x = "bye";

هل يتنافى ذلك مع Immutable ؟

* على الكومبايلر، لم يعيل معاملة، واعتبر التعديل على x مثل مرة تغيير المؤشر، ولم تتغير y.

String Methods

String one = "ahmadA";
String two = "ahmad";

① println(one.compareTo(two));

لما يبدأ صرف بحرف

مثل C

one > two , output > 0
two > one , output < 0
one == two , output 0

→ here +ve

رجع +عدد

return type → number

→ checking letter by letter

② println(one.equals(two));

تفحص المحتوى كاملاً (تفرق)

return type → boolean

(3) Println(one == two);

Check if they are directed to the same address.

* return type → boolean

(4) length method

int y = one.length();

method ; not property

(5) 1 char of a String

String y = "Khaled";

char x = y.charAt(2);

Print (x)

→ a

ex. Print ("Khaled".charAt(0));

→ K

allowed

(6) Capital letters

Print ln("Khaled".toUpperCase());

→ KHALED

(7) Small letters

Print ("KHaLEd".toLowerCase());

→ khaled

(8) equality ignoring capital & small

Print (one.equalsIgnoreCase(two));

.equals() مثل

يتجاهل الفرق بين الحرف الكبير والصغير

return type → boolean

(9) Compare ignoring ---

Print (one.compareToIgnoreCase(two));

returns a number

one > two +ve

two > one -ve

two = one 0

(10) Find index

String X = "Ahmad";

Print (X.indexOf('m'));

→ 2

يطبع اول تكرار للحرف

(11) Cut a String . Known start

String ~~x~~ = "Hello";

String y = x.substring(1);

num of char
not index

ex Print(y);

→ ello

(12) Cut a string, Known start, end

y = x.substring(1, 3);

indexes

included not included

ex Print(y)

→ el

ex Print(y.substring(1,1));

→ e

Print(x.substring(1,0)

→ error

(13) contains

Print("Hello".contains("el"));

→ return type → boolean

true

(14) concat

```
String x = "Hi";
String y = "Hi";

x = x.concat(y);

print(x);
```

→ Hi Hi

```
print(y.concat("Hello"));
```

→ Hi Hello

```
String a = "Hi";
String b = "Hi";
String c = a.concat(b);

print(a);
print(b);
print(c)
```

→ Hi
Hi
Hi Hi

```
c = a + b;
```

→ Hi
Hi
Hi Hi

Way (1)

Way (2)

ex
```
print("Hello" + "Hi");
```
→ HelloHi

ex
```
String x = "Hello" + "Hi";
print(x);
```
→ HelloHi

ex
```
String x = "a";
x += "a";
print(x)
```
→ aa

** 15 Convert string to array to edit it

```
String x = "Hello";
Char [] arr = x.toCharArray();
arr[3] = 'x';
x = new String(arr);
Println(X);
```
→ Helxo

(16) convert string to number

```
String value = "345";
int x = 6 + Integer.parseInt(value);
```

→ x = 351

```
value = "100.5"
double y = 10 + Double.parseDouble(value);
```

→ y = 110.5

(17) convert a number to string

```
string x = 10 + "";   } Way 1
String x = String.valueOf(10); } Way 2
```

[___ boolean, double, int ، تأخذ كل الأنواع

(18) Splitting تقطيع

```
String sentence = "this is my sentence";
```

(أولى) `String [] words = sentence.split(" ");`
تقطع عند space

↩ لا يمكن تخزينها في string

```
// / / / = sentence.split("e");
```
او
عند حرف e

```
m = sentence.split("e");
```
صنوع التخبير هكذا بيخ حرف space

string [] words = sentence.split(" e");

← هذا ليس تحذير بتنبيه (e) او ( )

ah ed ← هذا يقطع عند توفرها حصا قبل

⇒ ah
d

⇐ عدا ذلك ← لا يقطع

---

ex string sent = " ahmad ";

string [] words = sent.split("m");

→ ah
ed } الحرف تقطعه ولا تطبع

(*) ② او الحالة العادية بوضع عادي
للقطع عند ① هنا بين او ③ القطع عند الحروف (أ، إ، آ)

String [] words = sent.split("[?]");

String [] words = sent.split("[! ?]");
space ← عند ؟ ، ! ،

Note الارقام مثل الحرف ، لا تحتاج [] الا عند الكتابة

Note For-each: لا يعين طباعة من نوع array of strings

هل فقط int ؟

ex

```
String sent = "this is@my! sentence? Finally";
String [] words = sent.split("[@ !?]");

for (int i=0; i< words.length; i++)
    println(words[i]);
```

output →

this
is
my
sentence
Finally

Note ("[@]") , ("[?]") , ("[space]") , ~~("")~~ (" ");

---

(19) Matching (تطابق، مقارنة)

```
String sent = "this 1234 is my sent";

Print(sent.matches("this.*"));
```

→ true

```
Print(sent.matches("this\\d{3}.*"));
```

→ false

```
Print(sent.matches("this\\d{5}.*"));
```

→ true

ex String sent = "1234";

print(sent.matches("\\d{4}"));

→ true

print(sent.matches("\\d{3}"));

→ false , we didn't use .*

must be exactly 3 int.

print(sent.matches("\\d{3}.*"));

→ true

ex String sent = "1234"; → يبدأ بـ 3 أرقام

print(sent.matches("\\d{4}")); → false الفراغ يسبب

(20) string x = "Hello"; check last letter

print(x.endsWith("o"));

→ true

(21) check first letter

print(x.startsWith("H"));

→ true

(22) Replace regex regular expression

يتغير كل مرة لكن لا تأخذ

String sent = "Hello";

sent = sent.replace("e", "x");

print(sent);

→ Hxllo

recommended

replaceAll يغير كل حالات + regex تأخذ

replaceFirst يغير أول حالة فقط

(22) trim حقطع عالجوانب

```
String sent = "      Hi      ";
print( sent.trim() + "bye" );
```

→ Hi bye spaces تقطع

لازم ترجع

* reverse is in stringBuilder

String Builder.

* it's mutable (can be edited)

```
StringBuilder s = new StringBuilder ("ahmed");
```

Note Capacity means total size. (عدد الكاراكترات المحجوزة)
it's not as length(عدد الـ char

Rule by default, Capacity in the beginning = 16
Then it adds the string length to its capacity

So 16 + 5 = 21 (21 بيعد)

في حال مررنا string اثناء التعريف

```
print1n( s.capacity() );
print1n( s.length() );
```

→ 21
5

Rule — the default capacity is 16

Rule. use name.append("...") to add to StringBuilder

* يخزن متغيرات فيها

* عند مساواة الفتكس بـ StringBuilder اخرى يدخل عالشيء

ex

```
StringBuilder S1 = new StringBuilder("ahmed");
StringBuilder S2 = new StringBuilder();

S1.append("0");

S2 = S1.append("0");

Println(S1);
Println(S2);
Println(S1.equals(S2));
Println(S1 == S2);
```

output :

ahmed00
ahmed00
true
true

نفس الخرج

S2.append("0"); نفس جدل الطباعة

output

ahmed000
ahmed000
true
true

S1 يتم التعديل على الـ StringBuilder

mutable

```
Print(S1.capacity());
Print(S2.capacity());
```

→ 21 21

لان S2 تؤشر على S1

Rule When using argument constructor:

*) starts with capacity [16]

*) ~~it~~ it adds the capacity of the word

Note append() fills char in the capacity, ~~so~~ it doesn't change the capacity in general.

*) it changes the capacity if it is full.
~~it adds to capacity 1'st time =~~
~~it makes the capacity = length.~~
1'st time new capacity = (old+1)*2 ⟹ اذا اخترا ⟹ ~~بنزيد~~ بصير أعلى الحاجة

ex StringBuilder S = new StringBuilder("ahmed");

Println(S.capacity());
Println(S.length());

→ 21 (16+5)
→ 5

S.append("0");
println(S.capacity());
println(S.length());

→ 21
~~4~~6

the same

~~S = A~~ S = S.append ("01234567890123456780");

length = 24 ≤ ~~أكثر~~ أكثر الـ 21 الاصلي

Rule new capacity for first ~~time~~ time = (21+1)*2

```
println(s1.capacity());
println(s1.length());
```

→ 44
44

s1.append("----------") length=44 وصبح

s1.append("oo");

→ Print(s1.length());
Print(s1.capacity());

~~44~~ 46
46

length = capacity

كل ما نزيد احرف، نزيد بمقدارهم .

Rule when using no args constructor

String s2 = new StringBuilder();

Capacity = 16

ثم تنضيم عليه append

s2 = s1;

println(s2.capacity());
println(s2.length());

→ 46
46 لانه

// StringBuilder Methods

① name.append(" ___ ");

لا يعمل نسخة الحاجة الى اعادة string

لا يعوض اعادة string (يعمل على الاثنين)

② insert.

```
StringBuilder s1= new StringBuilder ("ahmed");
     ~      s2= ~          ~ ("Ali");
```

index

```
S1. insert(1, "by");
S2= S1. insert(1, "bye");
```

] — append نفس فكرة

Print(S1);

يضيف الكلام بعد ال index المذكور

→ ~~a bye hmed~~

ah bye med

③ stringBuilder s= ... ("ahmed") // Delete

```
S. delete(2,4);
Printh(s);
```

→ abd

2 داخلة

4 غير داخلة

stringBuilder 2 تحويل

(4) تغيير حرف

S. SetCharAt (1, 's');

Print(S);

→ as

لا يمكن ان نرجع stringBuilder كما

(5)

Str. Set length = new ___ ("Ahmad");

S. append("012345 ___);

Capacity = 44 حجم

S. SetLength(7);

Print(S);

Print(S. capacity());

→ ahmad01

44

(6) Delete capacity

*) بعد الامثال الاخيرة عندي زائد capacity

S. trimToSize();

Print(S. capacity());

→ 7

(*7) StringBuilder to string

StringBuilder x = new StringBuilder ("14");

String y = x. toString();

(6)

reverse

StringBuilder = _____ ("Ahmad");

S.reverse();

Print(s);

→ dambA