Pemrograman Berbasis Obyek

String

Oleh Politeknik Elektronika Negeri Surabaya

2017

Politeknik Elektronika Negeri Surabaya

Departemen Teknik Informatika dan Komputer

Konten

- Cara membuat obyek String
- Konstruktor String
- Sifat imutable String
- Equals method
- Reference String

The *String* Class

- Common string constructors:

```
String s1 = new String("immutable");
String s2 = "immutable";
```

- Class String berisi string yang tetap (immutable string).
- Sekali intance String dibuat maka isinya tidak bisa diubah.

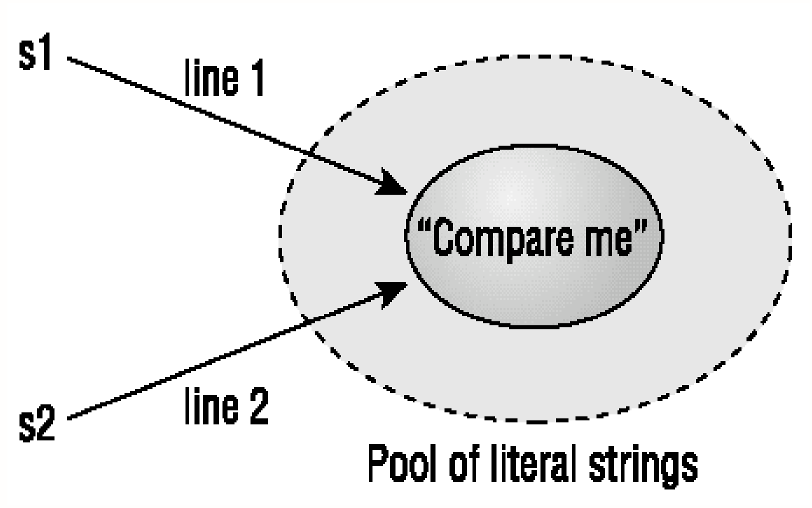
- Sebagaimana object java yang lain, object String bisa dibuat dengan menggunakan operator new (menggunakan constructor).
- Kelas String memiliki 13 constructors yang memungkinkan kita membuat object String dan menginisialisasi nilainya dengan menggunakan berbagai macam sumber data yang berbeda.
- Contoh:

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String helloString = new String(helloArray);
System.out.println(helloString);
```

The last line of this code snippet displays hello.

- Java mempunyai media penyimpanan literal string yang yang disebut “**pool**”.
- Jika suatu literal string sudah ada di pool, Java “tidak akan membuat copy lagi”.

Identical literals



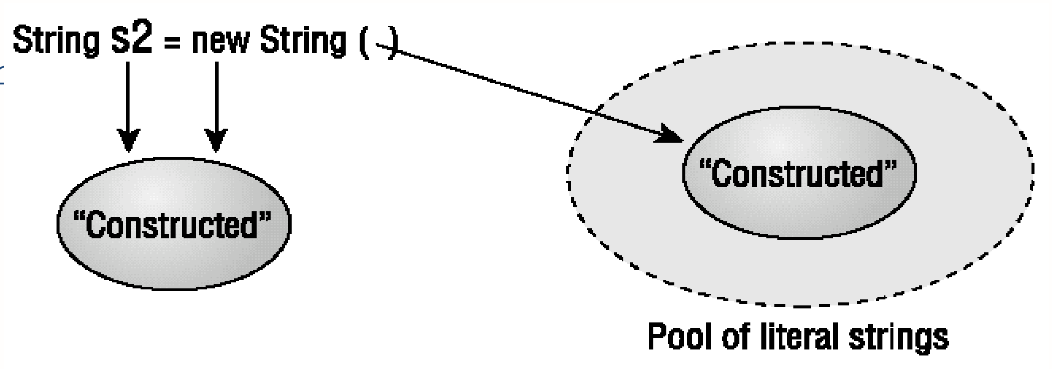
```
String s1 = "Compare me";
String s2 = "Compare me";
if (s1.equals(s2)) {
        // whatever
}
```

```
String s1 = "Compare me";
String s2 = "Compare me";
if (s1 == s2)) {
        // whatever
}
```

- Kedua potongan program diatas OK
- Contoh pertama membandingkan contentnya.
- Contoh kedua membandingkan referencesnya.

Explicitly calling the String constructor

```
String s2 = new Str
```



- Pada saat kompile → "Constructed" disimpan di pool tersendiri.
- Pada saat runtime → statement new String() dieksekusi, "Constructed" akan dibuatkan lagi di program space.
- Bila di pool sudah ada "Constructed", akan tetap dibuatkan program space.
- Supaya "Constructed" benar-benar disimpan di pool maka gunakan method intern().

equals() dan == untuk String

- Method equals() membandingkan content-nya
- == membandingkan alamatnya.

```
public class Equals{
    public static void main(String [] arg){
        String ja = "Ja";
        String va = "va";
        String java1 = ja + va ;
        String java = "Java";
        System.out.println("java.equals(java1)? " + java.equals(java1));
        System.out.println("java == java1? " + (java == java1));
    }
}
```

```
java.equals(java1)? true
java == java1? false
```

Variabel java1 dieksekusi pada saat runtime

String Length

- Methods used to obtain information about an object are known as *accessor methods*.
- One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object.
- After the following two lines of code have been executed, len equals 17:

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

Here is a short and inefficient program to reverse a palindrome string. It invokes the String method charAt(i), which returns the ith character in the string, counting from 0.

```
public class StringDemo {
    public static void main(String[] args) {
            String palindrome = "Dot saw I was Tod";
            int len = palindrome.length();
            char[] tempCharArray = new char[len];
            char[] charArray = new char[len];
```

```
palindrome.getChars(0, len, tempCharArray, 0);
```

```
            // put original string in an array of chars
            for (int i = 0; i < len; i++) {
                    tempCharArray[i] = palindrome.charAt(i);
            }

            // reverse array of chars
            for (int j = 0; j < len; j++) {
                    charArray[j] = tempCharArray[len - 1 - j];
            }

            String reversePalindrome = new String(charArray);        System.out.println(reversePalindrome);
    }
}
```

Concatenating Strings

- The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

- This returns a new string that is string1 with string2 added to it at the end.
- You can also use the concat() method with string literals, as in:

```
"My name is ".concat(“Ella");
```

- Strings are more commonly concatenated with the + operator, as in

```
"Hello," + " world" + "!"
```

- which results in

```
"Hello, world!"
```

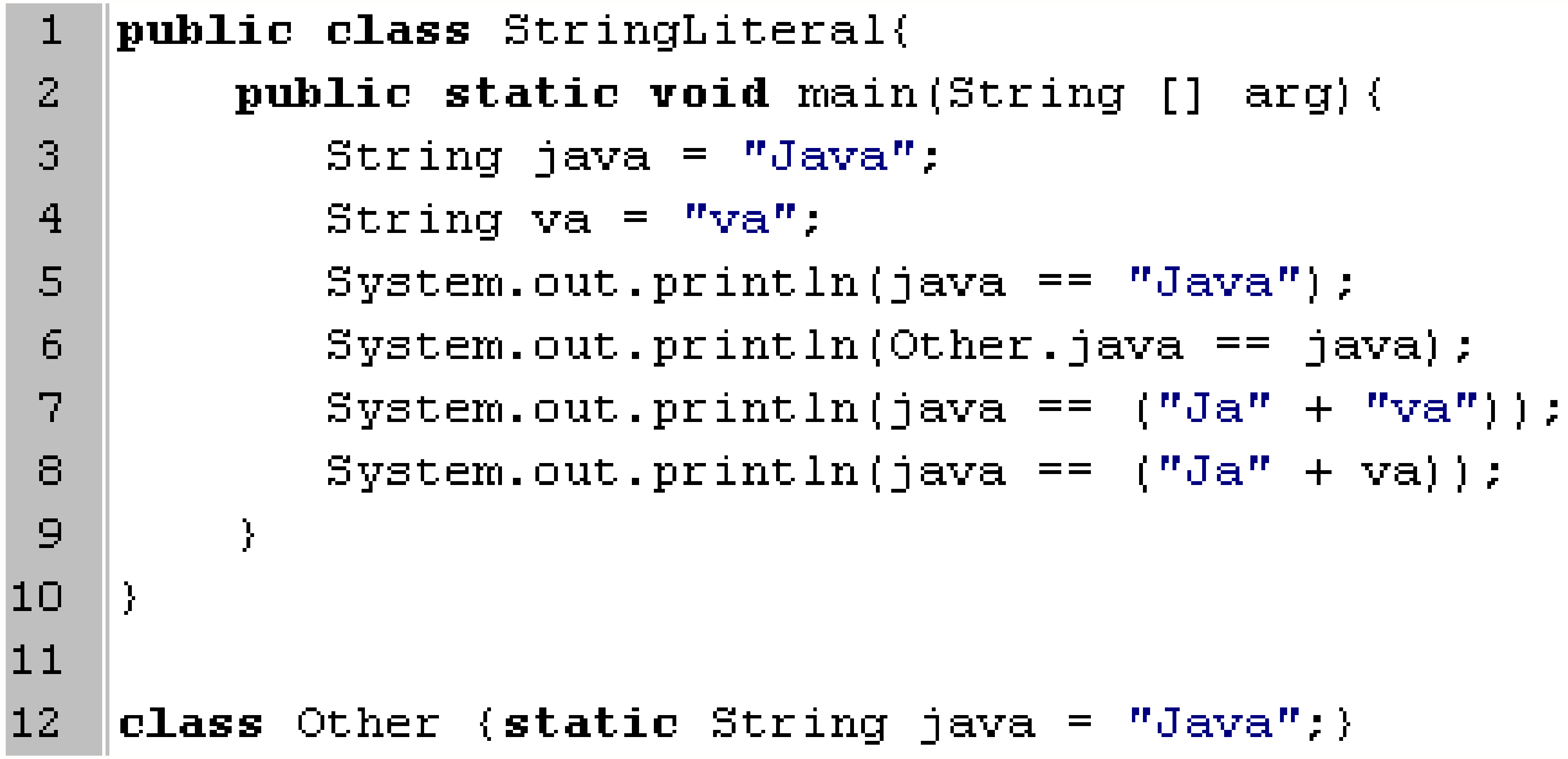
- The + operator is widely used in print statements.
- For example:

```
String string1 = "saw I was ";
System.out.println("Dot " + string1 + "Tod");
```

- which prints

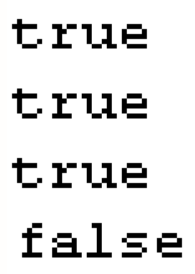
```
Dot saw I was Tod
```

- Such a concatenation can be a mixture of any objects.
- For each object that is not a String, its toString() method is called to convert it to a String.



```java
public class StringLiteral{
    public static void main(String [] arg){
        String java = "Java";
        String va = "va";
        System.out.println(java == "Java");
        System.out.println(Other.java == java);
        System.out.println(java == ("Ja" + "va"));
        System.out.println(java == ("Ja" + va));
    }
}

class Other {static String java = "Java";}
```



```
true
true
true
 false
```

String methods

- char charAt(int index): Returns the indexed character of a string, where the index of the initial character is 0.
- String concat(String addThis): Returns a new string consisting of the old string followed by addThis.
- int compareTo(String otherString): Performs a lexical comparison; returns an int that is less than 0 if the current string is less than otherString, equal to 0 if the strings are identical, and greater than 0 if the current string is greater than otherString.
- boolean endsWith(String suffix): Returns true if the current string ends with suffix; otherwise returns false.

String methods

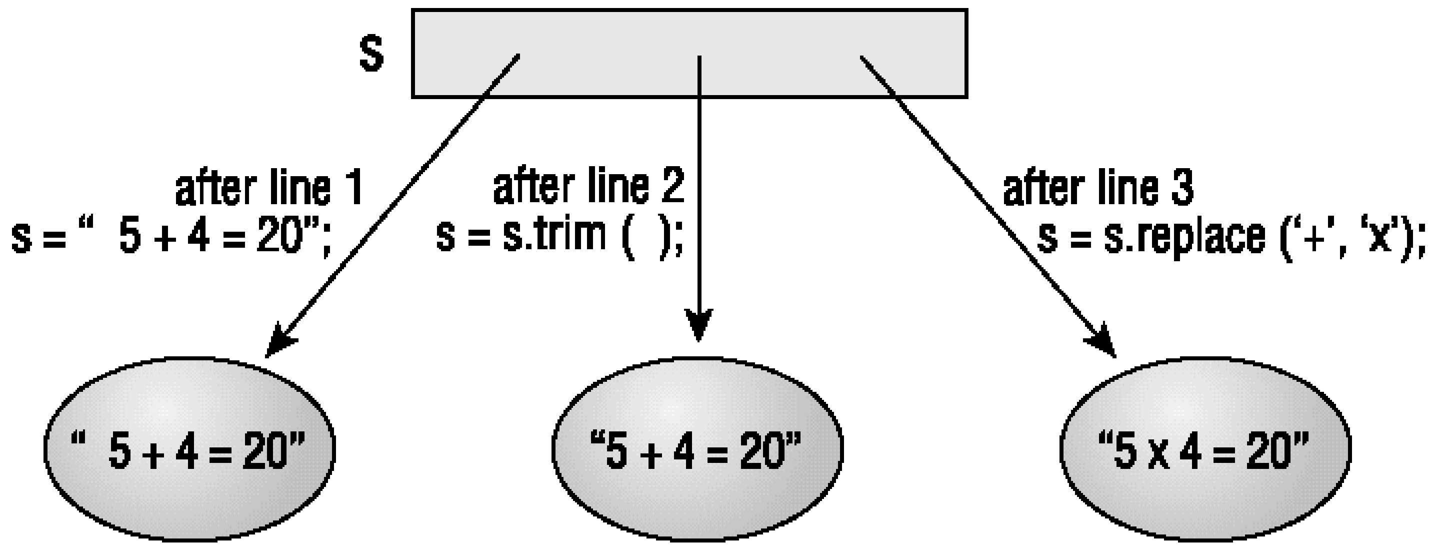
- boolean equals(Object ob): Returns true if ob instanceof String and the string encapsulated by ob matches the string encapsulated by the executing object.
- boolean equalsIgnoreCase(String s): Creates a new string with the same value as the executing object, but in lower case.
- int indexOf(int ch): Returns the index within the current string of the first occurrence of ch. Alternative forms return the index of a string and begin searching from a specified offset.
- int lastIndexOf(int ch): Returns the index within the current string of the last occurrence of ch. Alternative forms return the index of a string and end searching at a specified offset from the end of the string.

- int length(): Returns the number of characters in the current string.
- String replace(char oldChar, char newChar): Returns a new string, generated by replacing every occurrence of oldChar with newChar.
- boolean startsWith(String prefix): Returns true if the current string begins with prefix; otherwise returns false. Alternate forms begin searching from a specified offset.
- String substring(int startIndex): Returns the substring, beginning at startIndex of the current string and extending to the end of the current string. An alternate form specifies starting and ending offsets.
- String toLowerCase(): Creates a new string with the same value as the executing object, but in lower case.

- String toString(): Returns the executing object.
- String toUpperCase(): Converts the executing object to uppercase and returns a new string.
- String trim(): Returns the string that results from removing whitespace characters from the beginning and ending of the current string.

Trimming and replacing

```
1. String s = " 5 + 4 = 20";
2. s = s.trim(); // "5 + 4 = 20"
3. s = s.replace('+', 'x'); // "5 x 4 = 20"
```



The *StringBuffer* Class

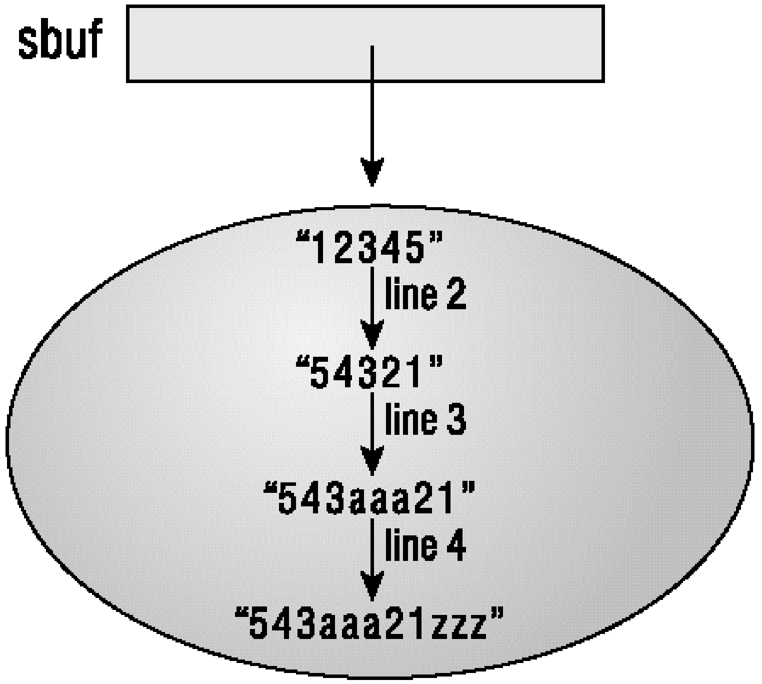
- represents a string that can be dynamically modified.
- Constructors:
  - `StringBuffer()`: Constructs an empty string buffer
  - `StringBuffer(int capacity)`: Constructs an empty string buffer with the specified initial capacity
  - `StringBuffer(String initialString)`: Constructs a string buffer that initially contains the specified string

StringBuffer methods

- StringBuffer append(String str): Appends str to the current string buffer. Alternative forms support appending primitives and character arrays; these are converted to strings before appending.
- StringBuffer append(Object obj): Calls toString() on obj and appends the result to the current string buffer.
- StringBuffer insert(int offset, String str): Inserts str into the current string buffer at position offset. There are numerous alternative forms.
- StringBuffer reverse(): Reverses the characters of the current string buffer.
- StringBuffer setCharAt(int offset, char newChar): Replaces the character at position offset with newChar.
- StringBuffer setLength(int newLength): Sets the length of the string buffer to newLength. If newLength is less than the current length, the string is truncated. If newLength is greater than the current length, the string is padded with null characters.

Modifying a string buffer

```
1. StringBuffer sbuf = new StringBuffer("12345");
2. sbuf.reverse(); // "54321"
3. sbuf.insert(3, "aaa"); // "543aaa21"
4. sbuf.append("zzz"); // "543aaa21zzz"
```



String Concatenation the Easy Way

- concat() method of the String class
- append() method of the StringBuffer class
- + operator.
- Example:

String Concatenation:

```
a + b + c
```

Java compiler treats as:

```
new StringBuffer().append(a).append(b).append(c).toString();
```

Tugas

1. Apakah perbedaan class String, StringBuffer dan StringBuilder?
2. Apakah yang dimaksud dengan sifat mutable dan immutable? Beri contoh!
3. Jelaskan operasi utama append dan insert yang dimiliki oleh StringBuffer!

1. Oracle Java Documentation, The Java™ Tutorials, https://docs.oracle.com/javase/tutorial/, Copyright © 1995, Oracle 2015.
2. Tita Karlita, Yuliana Setrowati, Rizky Yuniar Hakkun, Pemrograman Berorientasi Obyek, PENS-2012
3. Sun Java Programming, Sun Educational Services, Student Guide, Sun Microsystems, 2001.
4. John R. Hubbard, Programming With Java, McGraw-Hill, ISBN: 0-07-142040-1, 2004.
5. Patrick Niemeyer, Jonathan Knudsen, Learning Java, O'reilly, CA, ISBN: 1565927184, 2000.
6. Philip Heller, Simon Roberts, Complete Java 2 Certification Study Guide, Third Edition, Sybex, San Francisco, London, ISBN: 0-7821-4419-5, 2002.
7. Herbert Schildt, The Complete Reference, Java™ Seventh Edition, Mc Graw Hill, Osborne, ISBN: 978-0-07-163177-8, 2007