Strings & Wrapper Classes

CS102A Lecture 8

James YU
yujq3@sustech.edu.cn

Department of Computer Science and Engineering
Southern University of Science and Technology

Nov. 2, 2020

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Objective

- To create and manipulate strings.
  - Immutable character-string objects of class `String`.
  - Mutable character-string objects of class `StringBuilder`.
- To create and manipulate objects of class `Character`.
- Learn wrapper classes of primitive types.

Characters: Building blocks of Java programs

The primitive type `char`

- The `char` data type is a single 16-bit Unicode character.
  - `\u0000` – `\uffff`: 65536 characters, covering characters for almost all modern languages, and a large number of symbols.
- Programs often contain character literals (in single quotes).

```
1 char c1 = '\u0030';
2 char c2 = '\u0041';
3 char c3 = '\u4e2d';
4 char c4 = '\u56fd';
5 System.out.printf("%c %c %c %c", c1, c2, c3, c4);
```

```
0 A 中 国
```

The primitive type `char`

- The `char` data type is a single 16-bit Unicode character.
  - `\u0000` – `\uffff`: 65536 characters, covering characters for almost all modern languages, and a large number of symbols.
- Programs often contain character literals (in single quotes).

```java
char c1 = 'a';
char c2 = 97;
char c3 = '\u0061';
char c4 = 'a' + 1;
System.out.printf("%c %c %c %c", c1, c2, c3, c4);
```

```
a a a b
```

String

- A string is a sequence of characters
  - `"I like Java programming"`
- A string may include letters, digits and various special characters, such as +, -, *, / and $.
  - `"I \u2665 Java programming"`

Creating `String` objects

- `String` objects can also be created by using the `new` keyword and various `String` constructors.

```
String s1 = new String("hello world");
String s2 = new String(); // empty string (length is 0)
String s3 = new String(s1);
char[] charArray = {'h', 'e', 'l', 'l', 'o'};
String s4 = new String(charArray);
String s5 = new String(charArray, 1, 3); // string "ell"
```

String assignments

- A string may be assigned to a `String` reference.

```
1 String s = "hello world";
```

  - The statement initializes `String` variable `s` to refer to a `String` object that contains the string `"hello world"`.

```
1 String s2 = s;
```

  - The statement makes `s2` and `s` to refer to (sometimes we say “point to”, they mean the same thing) the same `String` object.

Comparing items

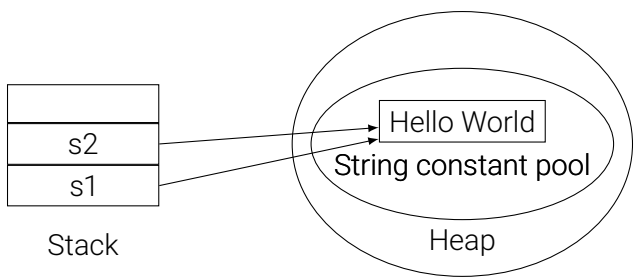
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
char c1 = 'a';
char c2 = 'a';
if (c1 == c2) {
  System.out.println("c1 and c2 are the same");
}
String s1 = new String("Hello");
String s2 = new String("Hello");
if (s1 == s2) {
  System.out.println("s1 and s2 are the same");
}
String s3 = "Hello";
String s4 = "Hello";
if (s3 == s4) {
  System.out.println("s3 and s4 are the same");
}
```

Creating `String` objects

- A string is an object of class `String`.
- `String` objects can be created by string literals (a sequence of characters in double quotes).

```
1 String s1 = "Hello World";
2 // no new objects will be created
3 String s2 = "Hello World";
```



Immutability

- In Java, `String` objects are immutable.
  - `String`s are constants; their values cannot be changed after they are created.
  - Because `String` objects are immutable, they can be shared safely.
- **Any modification creates a new `String` object**.

`String` methods

- `length` returns the length of a string (i.e., the number of characters).
- `charAt` helps obtain the character at a specific location in a string.
- `getChars` helps retrieve a set of characters from a string as a `char` array.
- These are instance methods that interact with the specific data of objects. Calling them requires an object reference.

The method `length`

```java
public class StringExamples {
  public static void main(String[] args) {
    String s1 = "hello world";
    System.out.printf("s1: %s", s1);
    System.out.printf("\nLength of s1: %d", s1.length());
  }
}
```

The method charAt

```java
public class StringExamples {
  public static void main(String[] args) {
    String s1 = "hello world";
    System.out.printf("s1: %s", s1);
    System.out.print("\nThe string reversed is: ");
    for(int count = s1.length() - 1; count >=0; count--) {
      System.out.printf("%c", s1.charAt(count));
    }
  }
}
```

The method `getChars`

- `getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`

```
1  public class StringExamples {
2    public static void main(String[] args) {
3      String s1 = "hello world";
4      char[] charArray = new char[5];
5      System.out.printf("s1: %s\n", s1);
6      s1.getChars(0, 5, charArray, 0);
7      for(char c : charArray) {
8        System.out.print(c);
9      }
10   }
11 }
```

Comparing `String`s

- When primitive-type values are compared with `==`, the result is `true` if both values are identical.

```
int a = 2, b = 2;
if (a == b) {
  System.out.println("a = b"); // prints a = b
}
```

- When references (memory addresses) are compared with `==`, the result is `true` if both references refer to the same object in memory.

```
String s1 = "Hello World";
String s2 = "Hello World";
if(s1 == s2) {
  System.out.println("s1 = s2"); // prints s1 = s2
}
```

Comparing `String`s

```
String s1 = "Hello World";
String s2 = s1 + "";
if(s1 == s2) {
  System.out.println("s1 = s2"); // prints s1 = s2?
}
```

- No. The condition will evaluate to `false` because the String variables `s1` and `s2` refer to two different `String` objects, although the strings contain the same sequence of characters.
- To compare the actual contents (or state information) of objects (strings are objects) for equality, a method must be invoked.

The method `equals`

- Method `equals` tests any two objects for equality – the strings contained in the two `String` objects are identical.

```
String s1 = "Hello World";
String s2 = s1 + "";
if(s1.equals(s2)) {
  System.out.println("s1 = s2"); // true
}
```

- Uses lexicographical comparison – it compares the integer Unicode values that represent each character in each `String`.

```
String s1 = "hello";
String s2 = "HELLO";
if(s1.equals(s2)) {
  System.out.println("s1 = s2"); // false
}
```

The method `equalsIgnoreCase`

- Method `equalsIgnoreCase` ignores whether the letters in each `String` are uppercase or lowercase when performing a comparison.

```
1 String s1 = "hello";
2 String s2 = "HELLO";
3 if(s1.equalsIgnoreCase(s2)) System.out.println("s1 = s2");
```

The method `compareTo`

- `compareTo` compares two strings (lexicographical comparison):
  - Returns `0` if the `String`s are equal (identical contents).
  - Returns a negative number if the `String` that invokes `compareTo` (`s1`) is less than the `String` that is passed as an argument (`s2`).
  - Returns a positive number if the `String` that invokes `compareTo` (`s1`) is greater than the `String` that is passed as an argument (`s2`).

```
1 String s1 = "hello";
2 String s2 = "HELLO";
3 int result = s1.compareTo(s2); // value of result?
```

Comparing strings

- What does it mean when we say a string `s1` is greater than another string `s2`?
  - When we sort last names, we naturally consider that “Jones” > “Smith”, because the letter ‘J’ comes before ‘S’ in the alphabet of 26 letters.
  - All characters in computers are represented as numeric codes. The characters form an ordered set (“a very large alphabet”).
  - When the computer compares `String`s, it actually compares the numeric codes of the characters in the `String`s.

The method `regionMatches`

- `regionMatches` compare portions of two `String`s for equality:
  - The first argument is the starting index in the `String` that invokes the method (`s1`).
  - The second argument is a comparison `String`.
  - The third argument is the starting index in the comparison `String`.
  - The last argument is the number of characters to compare between the two `String`s.
  - Returns `true` only if the specified number of characters are lexicographically equal.

```
1 String s1 = "Hello World";
2 String s2 = "hello world";
3 boolean result = s1.regionMatches(0, s2, 0, 5); // true or false?
```

The method `regionMatches`

- `regionMatches` is overloaded (it has a five-argument version):
  - When the first argument is `true`, the method ignores the case of the characters being compared.
  - The remaining arguments are identical to those described for the four-argument `regionMatches` method.

```
String s1 = "Hello World";
String s2 = "hello world";
boolean result = s1.regionMatches(true, 0, s2, 0, 5); // true
```

The method `startsWith` & `endsWith`

- The methods `startsWith` and `endsWith` determine whether a string starts or ends with the method argument, respectively.

```java
String s1 = "Hello World";
if(s1.startsWith("He")) {
  System.out.print("true"); // true
}

String s1 = "Hello World";
if(s1.startsWith("llo", 2)) {
  System.out.print("true"); // true
}

String s1 = "Hello World";
if(s1.endsWith("ld")) {
  System.out.print("true"); // true
}
```

Locating characters in `String`s

- `indexOf` locates the first occurrence of a character in a `String`.
  - If the method finds the character, it returns the character's index in the `String`; otherwise, it returns `-1`.
- Two-argument version of `indexOf`:
  - Take one more argument: the starting index at which the search should begin.

```
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.indexOf('c')); // 2
System.out.println(s.indexOf('$')); // -1
System.out.println(s.indexOf('a', 1)); // 13
```

Locating characters in `String`s

- `lastIndexOf` locates the last occurrence of a character in a `String`.
  - The method searches from the end of the `String` toward the beginning.
  - If it finds the character, it returns the character's index in the String; otherwise, it returns `-1`.
- Two-argument version of `lastIndexOf`:
  - The character and the index from which to begin searching backward.

```
1 String s = "abcdefghijklmabcdefghijklm";
2 System.out.println(s.lastIndexOf('c')); // 15
3 System.out.println(s.lastIndexOf('$')); // -1
4 System.out.println(s.lastIndexOf('a', 8)); // 0
```

Locating substrings in `String`s

- The versions of methods `indexOf` and `lastIndexOf` that take a `String` as the first argument perform identically to those described earlier except that they search for sequences of characters (or substrings) that are specified by their `String` arguments.

```
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.indexOf("def")); // 3
System.out.println(s.indexOf("def", 7)); // 16
System.out.println(s.indexOf("hello")); // -1
System.out.println(s.lastIndexOf("def")); // 16
System.out.println(s.lastIndexOf("def", 7)); // 3
System.out.println(s.lastIndexOf("hello")); // -1
```

Extracting substrings from `String`s

- `substring` methods create a new `String` object by copying part of an existing `String` object.
- The one-integer-argument version specifies the starting index (inclusive) in the original `String` from which characters are to be copied.
- Two-integer-argument version specifies the starting index (inclusive) and ending index (exclusive) to copy characters in the original `String`.

```
1 String s = "abcdefghijklmabcdefghijklm";
2 System.out.println(s.substring(20)); // hijklm
3 System.out.println(s.substring(3, 6)); // def
```

Concatenating `String`s

- `String` method `concat` concatenates two `String` objects and returns a new `String` object containing the characters from both original `String`s. The original `String`s to which `s1` and `s2` refer are not modified (recall that `String`s are immutable).

```
1 String s1 = "Happy ";
2 String s2 = "Birthday";
3 System.out.println(s1.concat(s2));
4 System.out.println(s1);
```

The method `replace`

- `replace` returns a new `String` object in which every occurrence of the first character argument is replaced with the second character argument. An overloaded version of method `replace` enables you to replace substrings rather than individual characters.

```
1 String s1 = "Hello";
2 System.out.println(s1.replace('l', 'L')); // HeLLo
3 System.out.println(s1.replace("ll", "LL")); // HeLLo
```

`String` case conversion methods

- `String` method `toUpperCase` returns a new `String` with uppercase letters where corresponding lowercase letters exist in the original.
- `String` method `toLowerCase` returns a new `String` object with lowercase letters where corresponding uppercase letters exist in the original.

```
1 String s1 = "Hello";
2 System.out.println(s1.toUpperCase()); // HELLO
3 System.out.println(s1.toLowerCase()); // hello
```

The method `trim`

- `trim` returns a new `String` object that removes all white-space characters at the beginning or end of the `String` on which trim operates.

```
1 String s1 = " spaces ";
2 System.out.println(s1.trim()); // prints "spaces"
```

The method `toCharArray`

- `toCharArray` creates a new character array containing a copy of the characters in the string.

```
String s1 = "hello";
char[] charArray = s1.toCharArray();
for(char c : charArray) {
  System.out.print(c);
}
```

Tokenizing `String`s

- When you read a sentence, your mind breaks it into tokens — individual words and punctuation marks that convey meaning to you.
- `String` method `split` breaks a `String` into its component tokens, separated from each other by *delimiters*, typically white-space characters such as space, tab, new line, carriage return (`\r`).

Tokenizing Strings

```
1 Scanner input = new Scanner(System.in);
2 System.out.println("
    Enter a sentence and press Enter");
3 String sentence = input.nextLine();
4 String[] tokens = sentence.split(" ");
5 System.out.printf("Number of tokens: %d\n"
    , tokens.length);
6 for(String token : tokens) {
7   System.out.println(token);
8 }
9 input.close();
```

```
Enter a sentence and press
Enter
This is a sentence with
seven tokens
Number of tokens: 7
This
is
a
sentence
with
seven
tokens
```

The method `valueOf`

- Every object in Java has a `toString` method that enables a program to obtain the object's String representation.
- Unfortunately, this technique cannot be used with primitive types because they do not have methods.
- Class `String` provides static methods that take an argument of any type and convert it to a `String` object.

Tokenizing `Strings`

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
boolean booleanValue = true;
char charValue = 'Z';
int intValue = 7;
long longValue = 10000000000L;
float floatValue = 2.5f;
double doubleValue = 33.3333; // no f suffix,
    double is default
char[] charArray = {'a', 'b', 'c', 'd', 'e', 'f'};
System.out.println(String.valueOf(booleanValue));
System.out.println(String.valueOf(charValue));
System.out.println(String.valueOf(intValue));
System.out.println(String.valueOf(longValue));
System.out.println(String.valueOf(floatValue));
System.out.println(String.valueOf(doubleValue));
System.out.println(String.valueOf(charArray));
```

```
true
Z
7
10000000000
2.5
33.3333
abcdef
```

Wrapper classes

- Java has 8 primitive types: `boolean`, `char`, `double`, `float`, `byte`, `short`, `int` and `long`.
- Java also provides 8 type-wrapper classes: `Boolean`, `Character`, `Double`, `Float`, `Byte`, `Short`, `Integer` and `Long` that enable primitive-type values to be treated as objects.
  - Be careful: not `Int` or `Char`.

`Character` class

- The class `Character` is the type-wrapper class for the primitive type `char`.
- `Character` provides methods (mostly static ones) for convenience in processing individual `char` values.
  - `isDigit(char c)`
  - `isLetter(char c)`
  - `isLowerCase(char c)`

`Character` class

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter a character and press Enter:");
String input = sc.next();
char c = input.charAt(0);

System.out.printf("is digit: %b\n", Character.isDigit(c));
System.out.printf("is identifier start: %b\n", Character.
    isJavaIdentifierStart(c));
System.out.printf("is letter: %b\n", Character.isLetter(c));
System.out.printf("is lower case: %b\n:", Character.isLowerCase(c));
System.out.printf("is upper case: %b\n", Character.isUpperCase(c));
System.out.printf("to upper case: %c\n", Character.toUpperCase(c));
System.out.printf("to lower case: %c\n", Character.toLowerCase(c));

sc.close();
```

`Character` class

```
Enter a character and press Enter:
A
is digit: false
is identifier start: true
is letter: true
is lower case: false
is upper case: true
to upper case: A
to lower case: a
```

```
Enter a character and press Enter:
8
is digit: true
is identifier start: false
is letter: false
is lower case: false
is upper case: false
to upper case: 8
to lower case: 8
```

`Character` object

```java
Character c1 = 'A';
Character c2 = new Character('A');

if (c1 == c2) {
  System.out.println("cc1 and cc2 are the same");
}

if (c1.equals(c2)) {
  System.out.println("cc1 and cc2 are the same");
}
```