Computer Programming I - CSC111

Chapter 2 – Basic Computation

Dr. Mejdl Safran
mejdl@ksu.edu.sa

Outline

- The class `String`
- Documentation and style

The class `String`

- We've used constants of type `String` already.

  `"Enter the amount to be changed: "`

- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

String constants and variables

- Declaring

```
String greeting;
 greeting = "Hello!";
```

 or

```
 String greeting = "Hello!";
```

 or

```
 String greeting = new String("Hello!");
```

- Printing

```
System.out.println(greeting);
```

Concatenation of strings and integers

```
String solution;
solution = "The answer is " + 42;
System.out.println (solution);
```

```
The answer is 42
```

Concatenation of strings and integers

```
System.out.println(2 + 5 + " Hello");
```

```
7 Hello
```

```
System.out.println("Hello " + 2 + 5);
```

```
Hello 25
```

String methods

- An object of the `String` class stores data consisting of a sequence of characters.
- Objects have methods as well as data
- The `length()` method returns the number of characters in a particular `String` object.

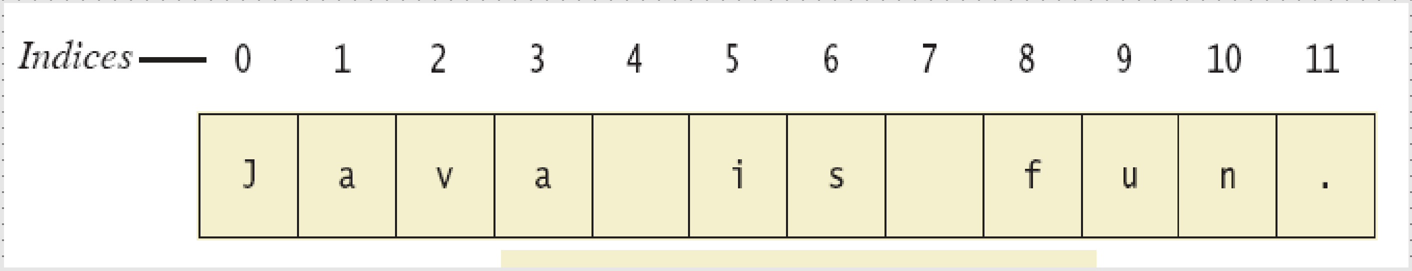
```
 String greeting = "Hello";
int n = greeting.length();
```

The method `length()`

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

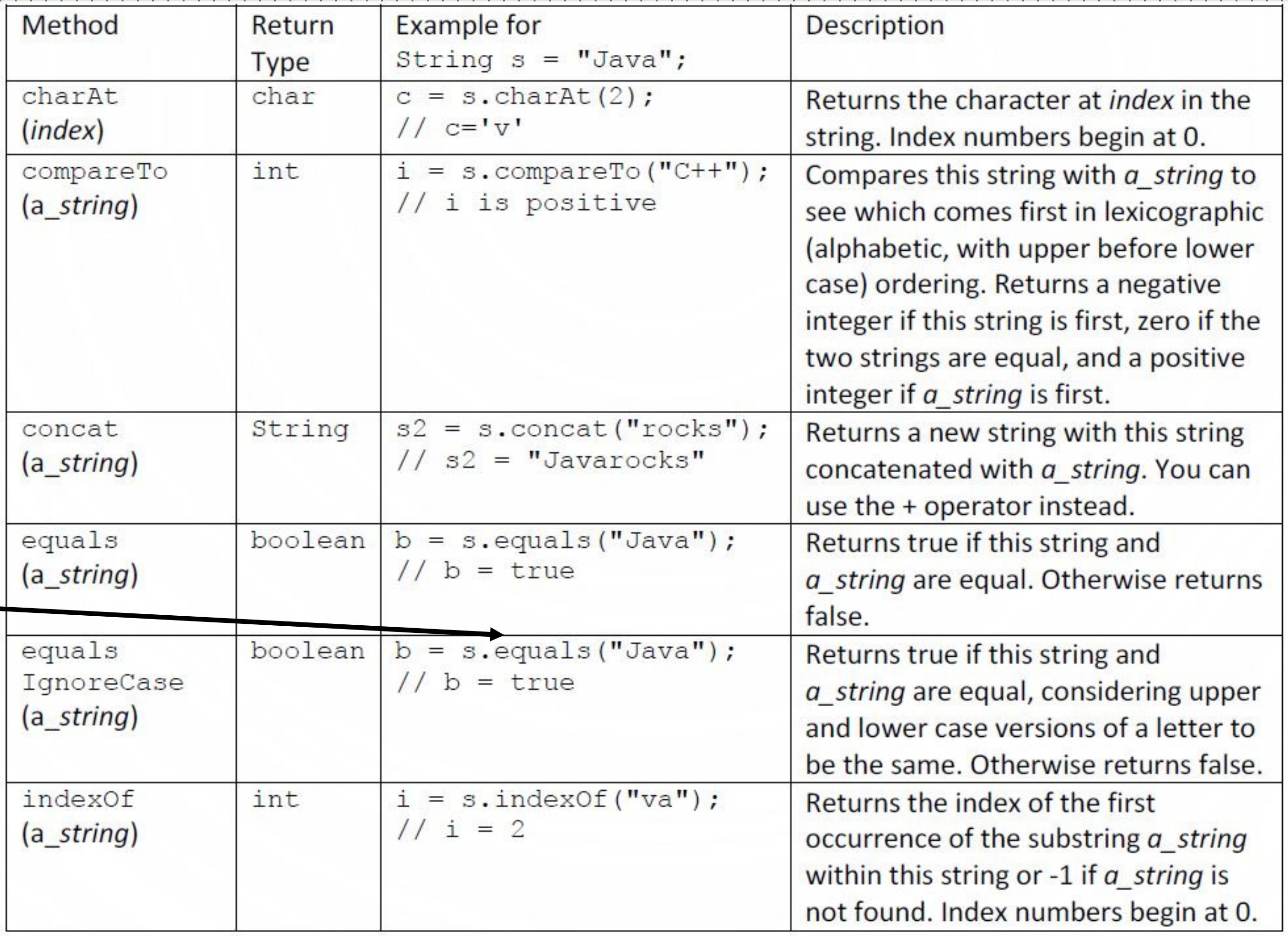
```
int count = command.length();
System.out.println("Length is " +
    command.length());
count = command.length() + 3;
```

`String` indices



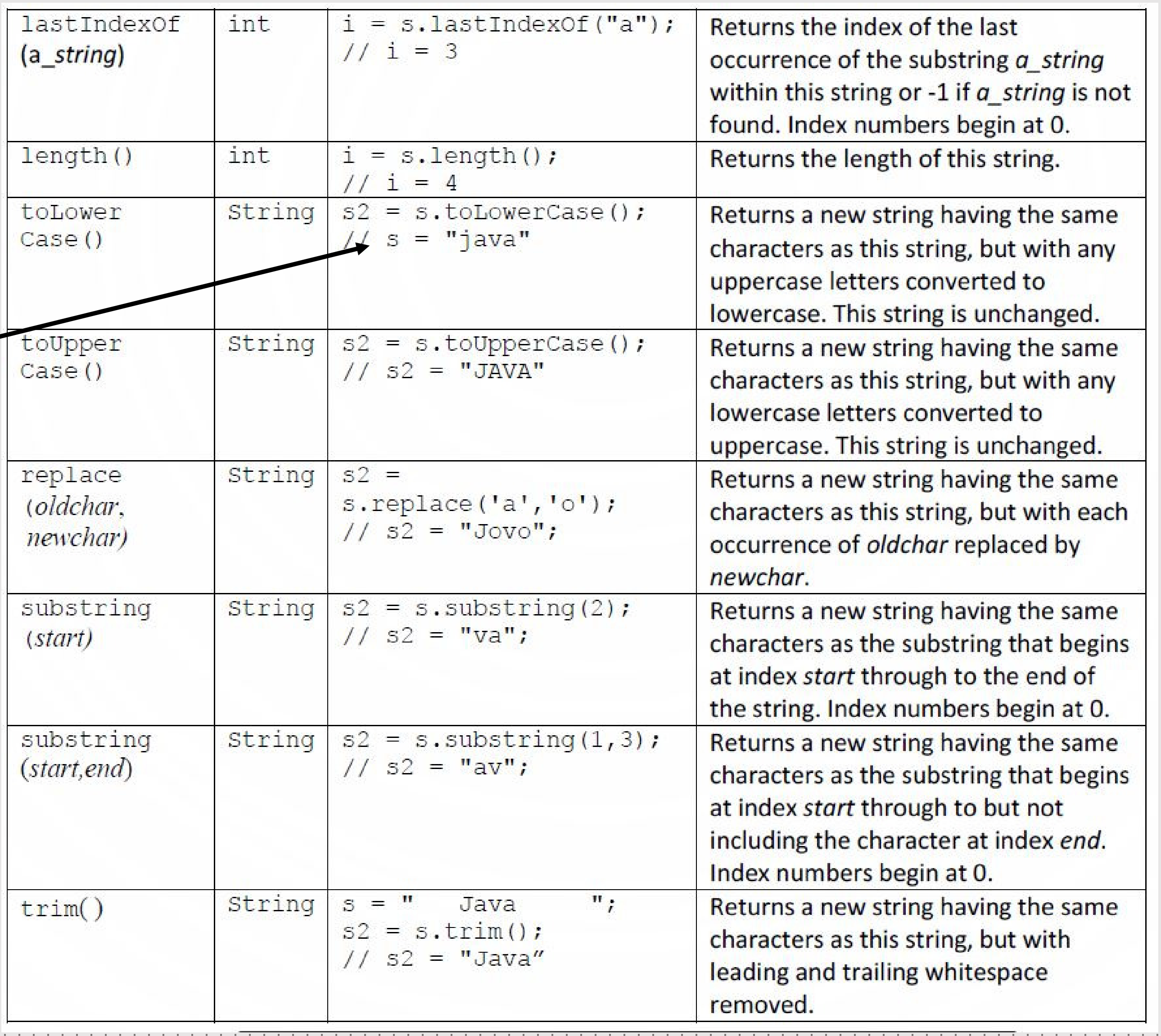
- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
- A position is referred to an *index.*
  - The `'f'` in `"Java is fun."` is at index 8.

`String` methods



Method	Return Type	Example for `String s = "Java";`	Description
`charAt` (*index*)	`char`	`c = s.charAt(2);` `// c='v'`	Returns the character at *index* in the string. Index numbers begin at 0.
`compareTo` (a_*string*)	`int`	`i = s.compareTo("C++");` `// i is positive`	Compares this string with *a_string* to see which comes first in lexicographic (alphabetic, with upper before lower case) ordering. Returns a negative integer if this string is first, zero if the two strings are equal, and a positive integer if *a_string* is first.
`concat` (a_*string*)	`String`	`s2 = s.concat("rocks");` `// s2 = "Javarocks"`	Returns a new string with this string concatenated with *a_string*. You can use the + operator instead.
`equals` (a_*string*)	`boolean`	`b = s.equals("Java");` `// b = true`	Returns true if this string and *a_string* are equal. Otherwise returns false.
`equals IgnoreCase` (a_*string*)	`boolean`	`b = s.equals("Java");` `// b = true`	Returns true if this string and *a_string* are equal, considering upper and lower case versions of a letter to be the same. Otherwise returns false.
`indexOf` (a_*string*)	`int`	`i = s.indexOf("va");` `// i = 2`	Returns the index of the first occurrence of the substring *a_string* within this string or -1 if *a_string* is not found. Index numbers begin at 0.

This should be equalsIgnoreCase, instead of equals



`lastIndexOf` (*a_string*)	`int`	`i = s.lastIndexOf("a");`
		`// i = 3`
			Returns the index of the last occurrence of the substring *a_string* within this string or -1 if *a_string* is not found. Index numbers begin at 0.

`length()`	`int`	`i = s.length();`
		`// i = 4`
			Returns the length of this string.

`toLower Case()`	`String`	`s2 = s.toLowerCase();`
		`// s = "java"`
			Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase. This string is unchanged.

`toUpper Case()`	`String`	`s2 = s.toUpperCase();`
		`// s2 = "JAVA"`
			Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase. This string is unchanged.

`replace` (*oldchar, newchar*)	`String`	`s2 =`
		`s.replace('a','o');`
		`// s2 = "Jovo";`
			Returns a new string having the same characters as this string, but with each occurrence of *oldchar* replaced by *newchar*.

`substring` (*start*)	`String`	`s2 = s.substring(2);`
		`// s2 = "va";`
			Returns a new string having the same characters as the substring that begins at index *start* through to the end of the string. Index numbers begin at 0.

`substring` (*start,end*)	`String`	`s2 = s.substring(1,3);`
		`// s2 = "av";`
			Returns a new string having the same characters as the substring that begins at index *start* through to but not including the character at index *end*. Index numbers begin at 0.

`trim()`	`String`	`s = "   Java     ";`
		`s2 = s.trim();`
		`// s2 = "Java"`
			Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

This should be s2, instead of s

Escape characters

- How would you print

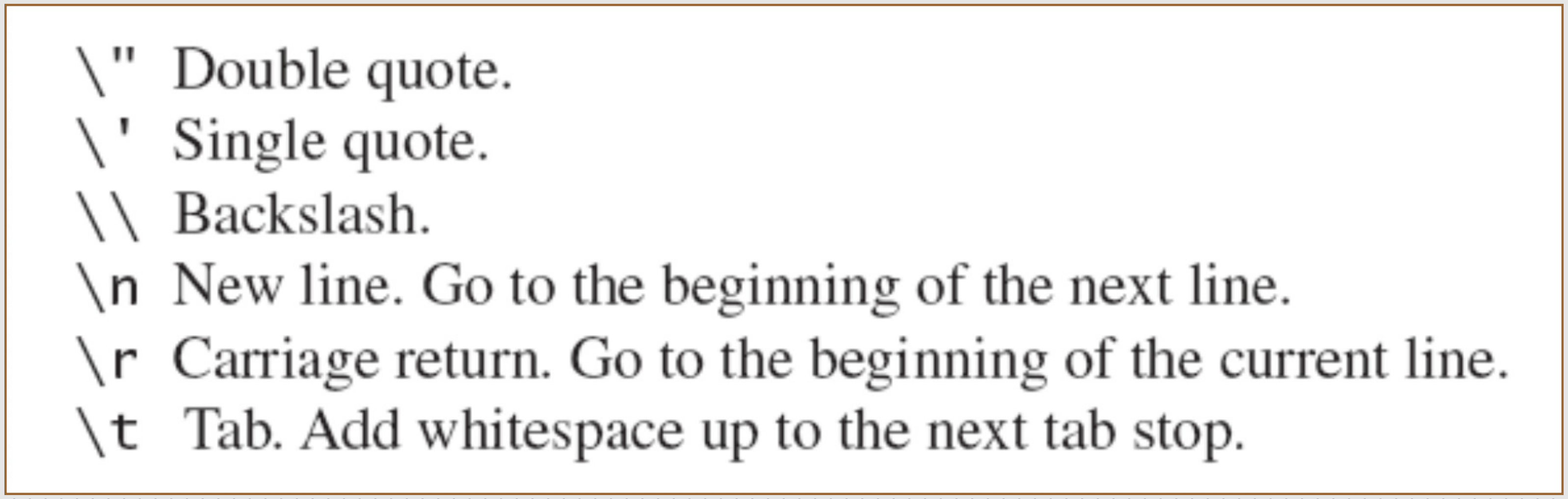
  `"Java" refers to a language.` ?

- The compiler needs to be told that the quotation marks (`"`) do not signal the start or end of a string, but instead are to be printed.

```
System.out.println(
"\"Java\" refers to a language.");
```

Escape characters

- Each escape sequence is a single character even though it is written with two symbols.



`\"`	Double quote.
`\'`	Single quote.
`\\`	Backslash.
`\n`	New line. Go to the beginning of the next line.
`\r`	Carriage return. Go to the beginning of the current line.
`\t`	Tab. Add whitespace up to the next tab stop.

Examples

```
System.out.println("abc\\def");
```

```
abc\def
```

```
System.out.println("new\nline");
```

```
new
line
```

```
char singleQuote = '\'';
System.out.println
   (singleQuote);
```

```
'
```

The empty string

- A string can have any number of characters, including zero.
- The string with zero characters is called the *empty* string.
- The empty string is useful and can be created in many ways including

```
String s3 = "";
```

.equals() vs. ==

- == tests for references equality (whether they are the same object)
- .equals() tests for value equality (whether they are logically “equal”)

- If you want to test whether two strings have the same value, you probably want to use .equals()

.equals() vs. ==

```
String a = "Test";
String b = "Test";
System.out.println(a==b);
```

true

```
String a = "Test";
String b = "Test";
System.out.println(a.equals(b));
```

true

```
String a = "Test";
String b = new String("Test");
System.out.println(a==b);
```

false

```
String a = "Test";
String b = new String("Test");
System.out.println(a.equals(b));
```

true

So using .equals() is always better.
A good example will be discussed when we study nextLine().

Back to Scanner methods

Method	Example
nextByte( )	byte b = input.nextByte( );
nextShort( )	short s = input.nextShort( );
nextInt( )	int i = input.nextInt( );
nextLong( )	long l = input.nextLong( );
nextFloat( )	float f = input.nextFloat( );
nextDouble( )	double d = input.nextDouble( );
next()	String str = input.next();
nextLine()	String str = input.nextLine();

next() vs. nextLine()

- `next()` reads the input only till the space. It can't read two words separated by space.
- `nextLine()` reads input including space between the words (that is, it reads till the end of the line).

nextLine() method caution

```java
int age;
String name;
Scanner keyboard = new Scanner(System.in);

System.out.println("Enter your name: ");
name = keyboard.nextLine();

System.out.println("Enter your age: ");
age = keyboard.nextInt();

System.out.println("Your name is: " + name);
System.out.println("Your age is: " + age);
```

Output

```
Enter your name:
Mohammad
Enter your age:
23
Your name is: Mohammad
Your age is: 23
```

nextLine() method caution

```java
int age;
String name;
Scanner keyboard = new Scanner(System.in);

System.out.println("Enter your age: ");
age = keyboard.nextInt();

System.out.println("Enter your name: ");
name = keyboard.nextLine();

System.out.println("Your name is: " + name);
System.out.println("Your age is: " + age);
```

Output

```
Enter your age:
23
Enter your name:
Your name is:
Your age is: 23
```

nextLine() method caution - solution

```java
int age;
String name;
Scanner keyboard = new Scanner(System.in);

System.out.println("Enter your age: ");
age = keyboard.nextInt();

keyboard.nextLine();

System.out.println("Enter your name: ");
name = keyboard.nextLine();

System.out.println("Your name is: " + name);
System.out.println("Your age is: " + age);
```

Output

```
Enter your age:
23
Enter your name:
Mohammad
Your name is: Mohammad
Your age is: 23
```

What is the output?

```java
int age, score;
String name;
Scanner keyboard = new Scanner(System.in);

age = keyboard.nextInt();
name = keyboard.nextLine();
score = keyboard.nextInt();

System.out.println("Your name is: " + name);
System.out.println("Your age is: " + age);
System.out.println("Your score is: " + score);
```

Output

```
33
Mohammad
Exception in thread "main"
java.util.InputMismatchException
at
java.util.Scanner.throwFor(Unknown
Source)
at java.util.Scanner.next(Unknown
Source)
at java.util.Scanner.nextInt(Unknown
Source)
at java.util.Scanner.nextInt(Unknown
Source)
at Test1.main(Test1.java:15)
```

== with nextLine()

```java
String s1 = "boys", s2;
Scanner keyboard = new Scanner(System.in);

System.out.println("Enter your text here: ");
s2 = keyboard.nextLine();

System.out.println(s1 == s2);
```

Output

```
Enter your text here:
boys
false
```

Documentation and style

- Meaningful Names
- Comments (*already covered*)
- Indentation

Documentation and style

- Most programs are modified over time to respond to new requirements.
- Programs which are easy to read and understand are easy to modify.
- Even if it will be used only once, you have to read it in order to debug it .

Meaningful variable names

- A variable's name should suggest its use.
- Observe conventions in choosing names for variables.
  - Use only letters and digits.
  - "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  - Start variables with lowercase letters.
  - Start class names with uppercase letters.

Indentation

- Indentation should communicate nesting clearly.
- A good choice is four spaces for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.

Indentation

- Indentation does not change the behavior of the program.
- Proper indentation helps communicate to the human reader the nested structures of the program