UNSW Business School
Information Systems and Technology Management

INFS1609/2609 Fundamentals of Business Programming

Lecture 3
Elementary Programming

yenni.tim@unsw.edu.au

INFS1609/2609 Fundamentals of Business Programming

Topics for this week:

- Java numeric primitive data types: `byte`, `short`, `int`, `long`, `float`, and `double`
- Operators, increments, decrements
- Cast the value of one type to another type
- The character data type and operations
- The `String` type
- Common programming errors related to data types and operation

Main references
- *Textbook: Chapter 2 and Chapter 4*
- *Other online references posted on Ed*

INFS1609/2609 Fundamentals of Business Programming

Numerical Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Numerical Data Types

A recap on the methods of Scanner

Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

An important point to note

**Integer Division
5 / 2 yields an integer 2

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

The Remainder Operator

- or the modulus operator
- Returns the **remainder** of two numbers
- `5 % 2` yields 1 (the remainder of the division)

A useful operator to determine whether a number is even or odd!

Let's look at a coding example on Ed

Number Literals

A *literal* is a value that **appears directly** in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;

long x = 1000000;

double d = 5.0;
```

Integer Literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable
- A **compilation error** would occur if the literal were too large for the variable to hold
- For example, `byte b = 1000` would cause a compilation error, because 1000 cannot be stored in a variable of the `byte` type

float and double Literals

- Floating-point literals are written with a **decimal point**. By default, a floating-point literal is treated as a `double` type value
- For example, 5.0 is considered a `double` value, not a `float` value
- In Java, you can make a number a `float` by appending the letter `f` or `F`, and make a number a `double` by appending the letter `d` or `D`

float and double Literals

The `double` type values are more accurate than the `float` type values. For example:

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays **`1.0 / 3.0 is 0.3333333333333333`**

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays **`1.0F / 3.0F is 0.33333334`**

7 digits

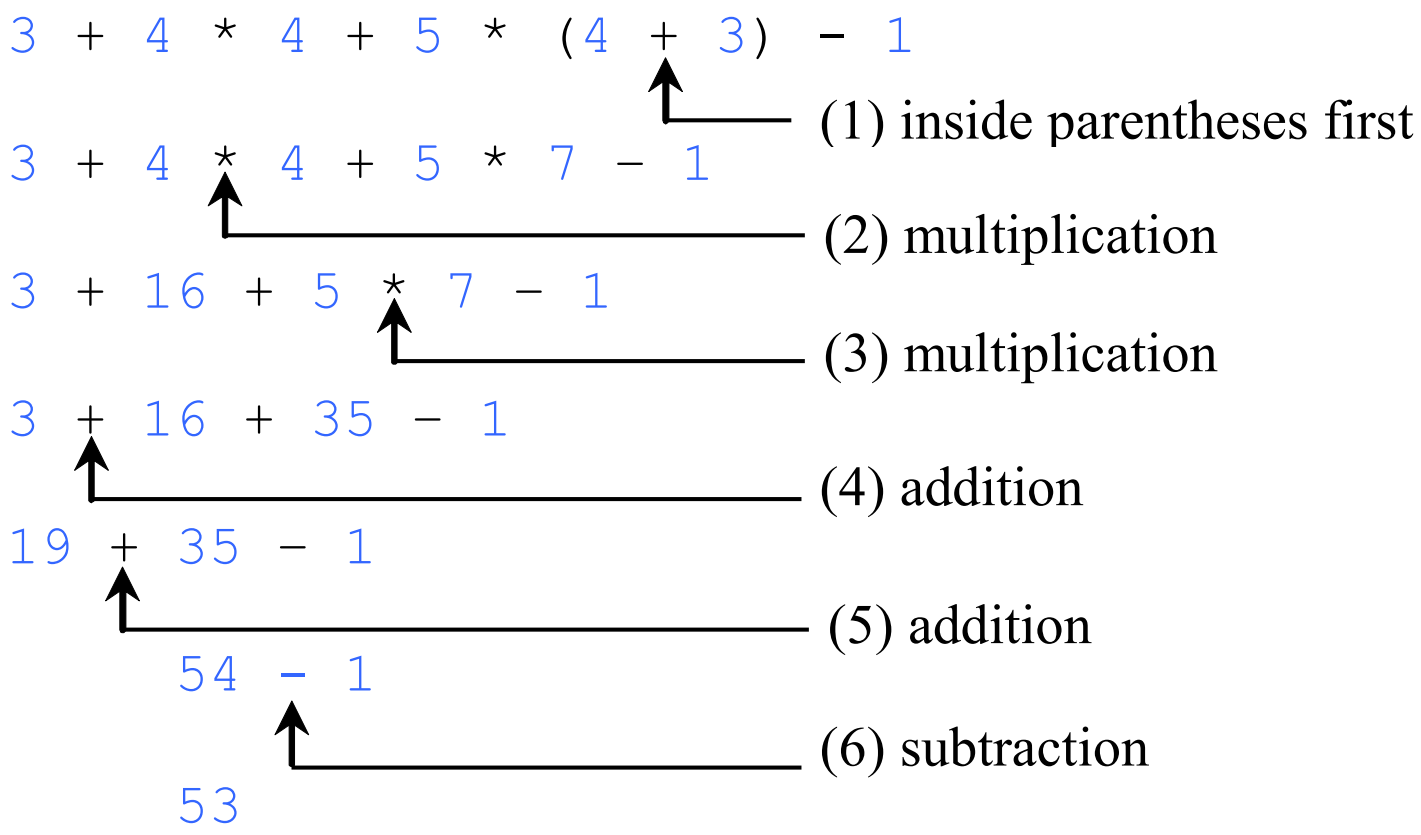
Arithmetic Expressions

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

```
(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

Tracing/evaluating an expression in Java



Let's practice some coding!

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

Hint. you have to write:

```
celsius = (5.0 / 9) * (fahrenheit - 32)
```

Answer: FahrenheitToCelsius

https://liveexample.pearsoncmg.com/html/FahrenheitToCelsius.html

Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

Increment and Decrement Operators

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

Questions?

Numeric Type Conversion

Primitive data types

- A type predefined by Java and is named by a reserved keyword
- We have eight primitive data types: _____
  - `byte, short, int, long, float, double, boolean, char`

Note that we **do not** use the `new` keyword to initialize a variable of primitive type – it is used for object

Numeric Type Conversion

Primitive data types

To understand each primitive data types, you need to understand 3 things:

- The set of possible **values**
- The set of possible **operations**
- The set of **literals** (**symbols** of the language that define the **values** of the data types, e.g., 10, 3.14, ‘A’, true)

Numeric Type Conversion

For example

The data type `int`

Type	`int`	
Dimension	32 bit (4 byte)	
Domain	the set of integer numbers in the interval $[-2^{31}, +2^{31} - 1]$ (more than 4 billion values)	
Operations	`+`	sum
	`-`	difference
	`*`	product
	`/`	integer division
	`%`	rest of the integer division
Literals	sequences of digits denoting values of the domain (e.g., 275930)	

Numeric Type Conversion

Implicit casting

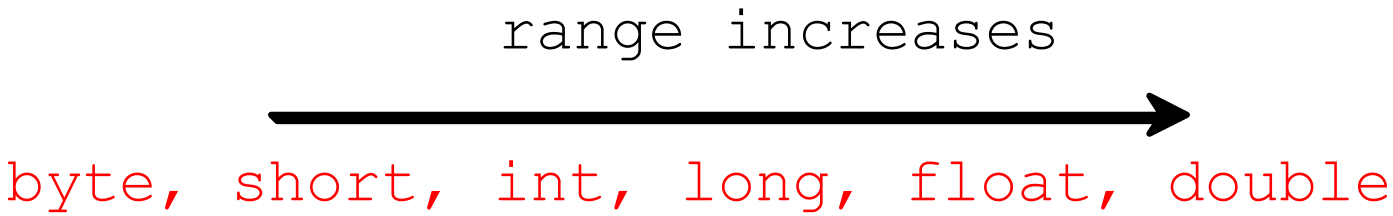
```
double d = 3;
```
(type widening)

Explicit casting

```
int i = (int)3.0;
```
(type narrowing)

```
int i = (int)3.9;
```
(Fraction part is truncated)

Numeric Type Conversion

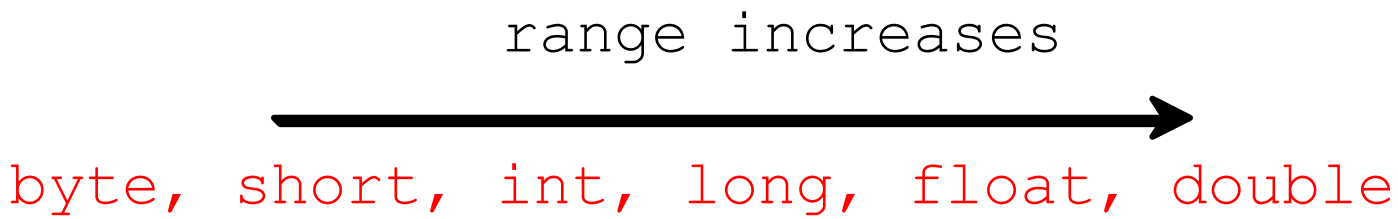


"Automatic" implicit casting

officially called as the **Assignment Conversion**

- Assign a value to a numeric variable whose type supports **a larger range of values** (widening a type)
- E.g., you can assign a `long` to a `float` variable

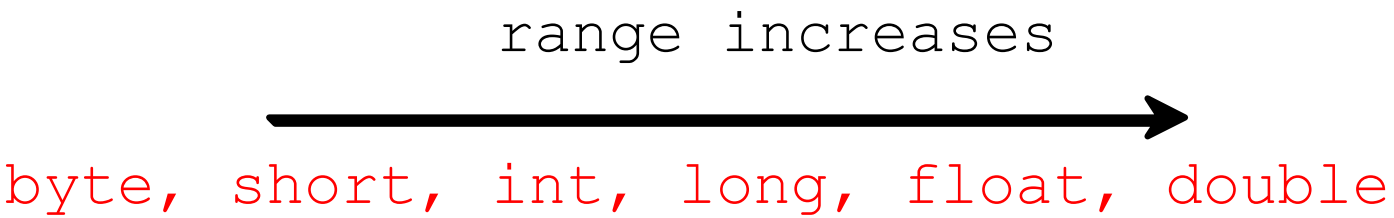
Numeric Type Conversion



Explicit type casting

- Narrowing a type – casting a type with a large range to a type with a smaller range
- Syntax: specify the target type in parentheses ( ), followed by the variable's name
- For example: `(int)1.7`

Numeric Type Conversion



Conversion rules

1. If one of the operands is `double`, the other is converted into `double`
2. Otherwise, if one of the operands is `float`, the other is converted into `float`
3. Otherwise, if one of the operands is `long`, the other is converted into `long`
4. Otherwise, both operands are converted into `int`

Numeric Type Conversion

What are the output?

```
System.out.println((double) 1 / 2);

System.out.println(1 / 2);
```

The first statement displays 0.5 because 1 is cast to 1.0, then divided by 2

Numeric Type Conversion

Note:

Casting does not change the variable being cast

For example, the variable `d` is not changed after casting in the following code:

```
double d = 4.5;
int i = (int)d; //i = 4 but d is still 4.5
```

INFS1609/2609 Fundamentals of Business Programming

Let's see how we can make use of casting to perform some useful tasks:

Write a program that displays the sales tax with two digits after the decimal point

SalesTax

https://liveexample.pearsoncmg.com/codeanimation/SalesTax.html

The Character Data Type and Operations

- A character data type represents a **single** character
- A character literal is enclosed in **single** quotation marks

```
char letter = 'A';
Char numChar = '4';
```

The Character Data Type and Operations

- As discussed in your first lecture, computers use binary numbers internally. A character is also stored in a computer as a sequence of 0s and 1s
- Mapping a character to its binary representation is called **encoding**

Think: where do you usually see encoding in action?

The Character Data Type and Operations

Unicode and ASCII code

```
char letter = 'A'; //ASCII
char letter = '\u0041'; //Unicode

char numChar = '4'; //ASCII
char numChar = '\u0034'; //Unicode
```

The Character Data Type and Operations

Looking at the table... what are the output for the following statements?

Characters	Code Value in Decimal	Unicode Value
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

```
int test = 65;
System.out.println((char)test);
```

The Character Data Type and Operations

Note: The increment and decrement operators can also be used with the char variables

- The operators will get the next, or preceding, Unicode character
- For example, the following statement displays the character `b`

```
char ch = 'a';
System.out.println(++ch);
```

How about Special Characters?

Suppose you want to print a double quote in your output. Can you write a statement like this?

```
System.out.println("My students say: "Java is fun!"");
```

Escape Sequences for Special Characters

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
`\b`	Backspace	`\u0008`	8
`\t`	Tab	`\u0009`	9
`\n`	Linefeed	`\u000A`	10
`\f`	Formfeed	`\u000C`	12
`\r`	Carriage Return	`\u000D`	13
`\\`	Backslash	`\u005C`	92
`\"`	Double Quote	`\u0022`	34

Casting between `char` and numeric types

```
int i = 'a'; // Same as int i = (int)'a';
```

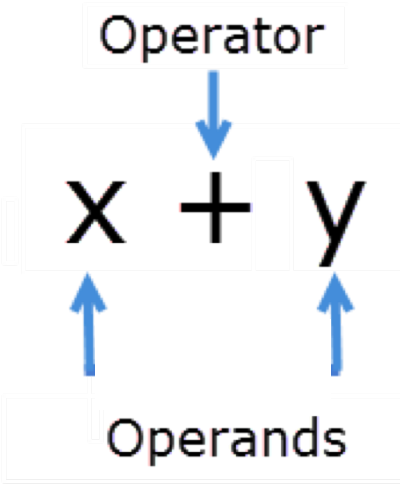
```
char c = 97; // Same as char c = (char)97;
```

Think about what is the output and test it on Netbeans!

Numeric Operators with the `char` type

- A `char` operand is **automatically** cast into a number if the other operand is a number or a character
- If the other operand is a String, the character is concatenated with the String

```
int i = '2' + '3';
System.out.println("i is: " + i);

//i is 101 because 2 is 50 and 3 is 51
```



Numeric Operators with the `char` type

```
int j = 2 + 'a'; //(int)'a' is 97
System.out.println("j is: " + j); //j is 99


System.out.println(j + " is the Unicode for
character " + (char)j);
//j is the Unicode for character c
```

Questions?

The String type

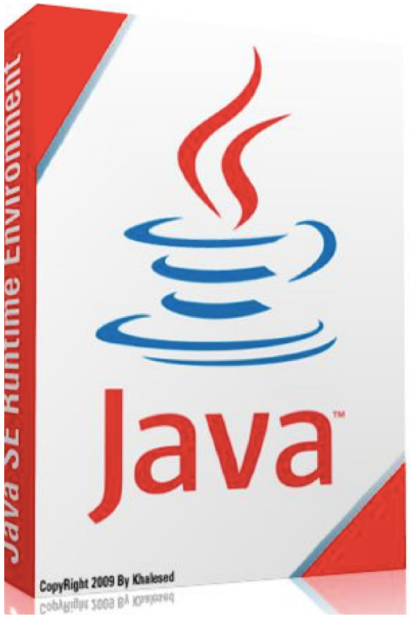
- String is actually a predefined class in the Java library just like the System class and Scanner class
- The String type is not a primitive type. It is known as a *reference type*
- Reference type will be thoroughly discussed later
- For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, how to concatenate strings, and to perform simple operations for strings

Simple methods for String Objects

Method	Description
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

INFS1609/2609 Fundamentals of Business Programming

A quick look at Oracle Java Docs



The String type

- Strings are objects in Java
- The methods in the preceding table can only be invoked from a specific string instance
- The syntax to invoke an instance method is

```
referenceVariable.methodName(arguments)
```

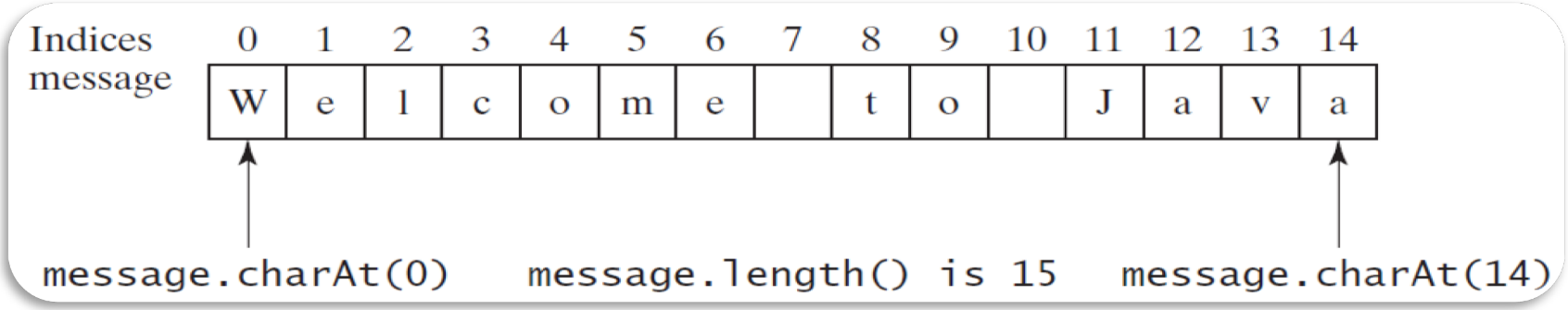
The String type

Working with the .length() method

```
String message = "Welcome to Java";
System.out.println("The length of " + message
+ " is " + message.length());
```

Getting characters from a String

Working with the .charAt() method




```
String message = "Welcome to Java";
System.out.println("The first character
in message is " + message.charAt(0));
```

Converting Strings

`"Welcome".toLowerCase()` returns a new string `welcome`

`"Welcome".toUpperCase()` returns a new string `WELCOME`

`"  Welcome  ".trim()` returns a new string `Welcome`

String Concatenation

```
String s3 = s1.concat(s2); or String s3 = s1 + s2;

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes
SupplementB
```

Reading Strings from the console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated
by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Reading a Character from the console

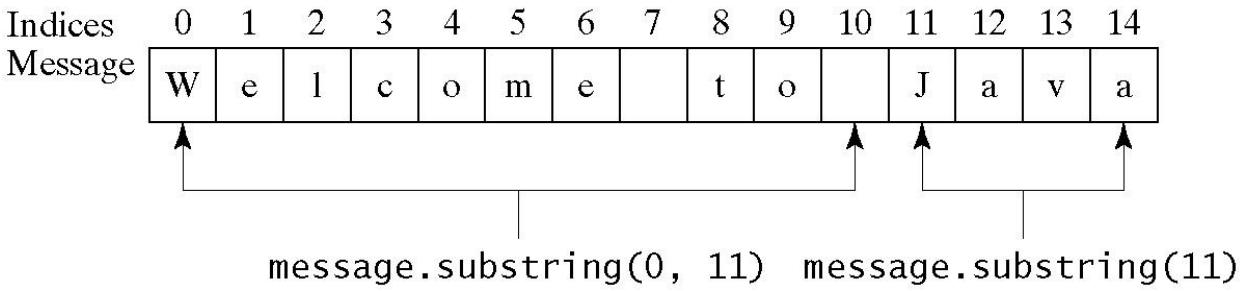
```
Scanner input = new Scanner(System.in);

System.out.print("Enter a character: ");

String s = input.nextLine();
char ch = s.charAt(0);

System.out.println("The character entered is " + ch);
```

Comparing Strings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

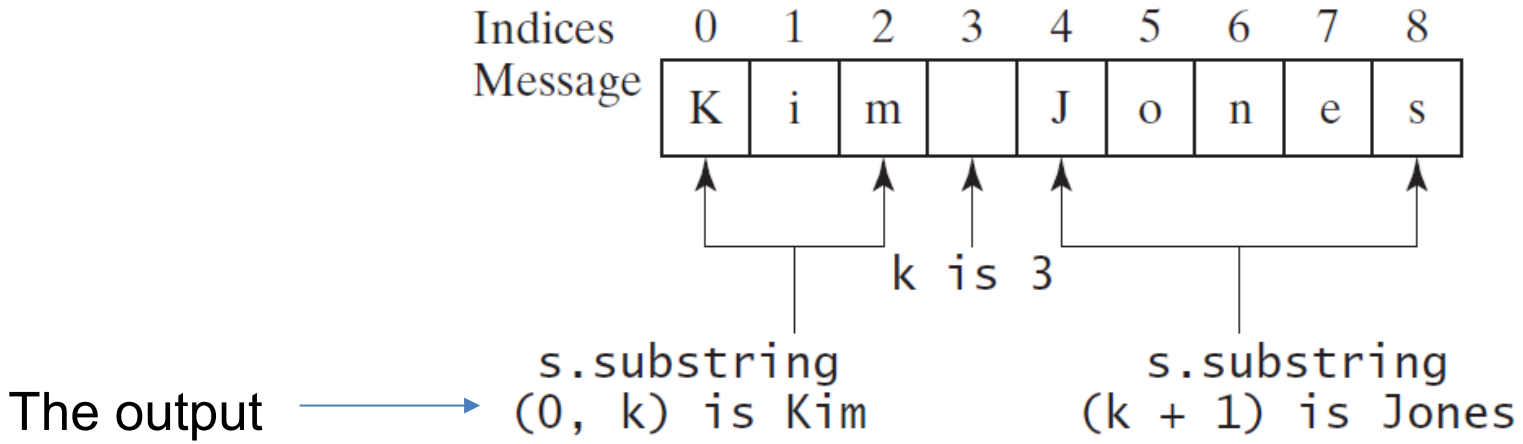
Obtaining substrings

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.



Obtaining substrings

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Conversion between Strings and numbers

```
String intString = "123";
int intValue = Integer.parseInt(intString);
```


parseInt is a method in the **Integer** class

Try to perform `parseInt` with a **non-numerical** String! What is the output?

Reflection

What have you learned today?

Up next!

- To declare boolean variables and write boolean expressions using relational operators
- To implement selection control using if, if-else and nested if statements
- To combine conditions using logical operators (&&, ||, and !)
- To implement selection control using switch statements

Thank you!