



# Java Foundations

4-3

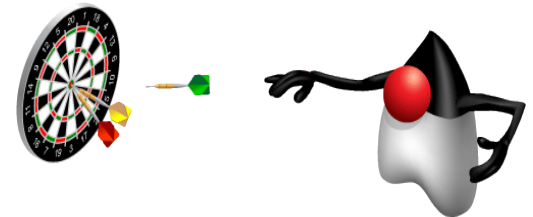
The `String` Class



# Objectives

This lesson covers the following objectives:

- Locate the `String` class in the Java API documentation
- Understand the methods of the `String` class
- Compare two `String` objects lexicographically
- Find the location of a substring in a `String` object
- Extract a substring from a `String` object



# Topics

- Getting Started with Strings
- `String` Class Documentation
- `String` Class Methods
- Concatenating Strings
- Comparing Strings

What is a  
Method?

The `import`  
Declaration  
and Packages

The `String`  
Class

The `Random`  
Class

The `Math`  
Class



Section 4

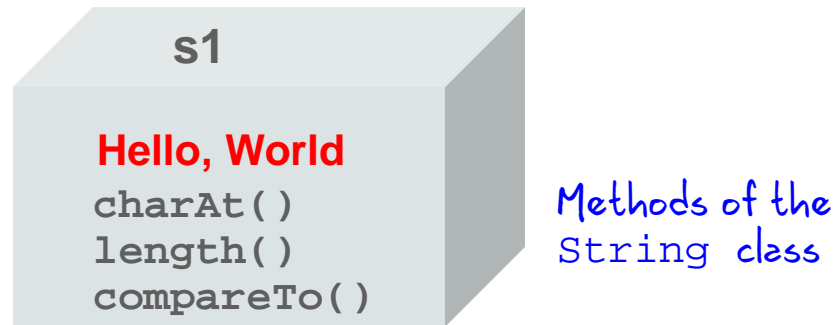
# What's a String?

- A string is a sequence of characters including alphabet letters, special characters, and white space.
- For example:
  - “How are you?” is a string that contains letters, white space, and a special character ('?').
- In Java, strings are not a primitive data type.
- Instead, they are objects of the `String` class.

# Representing Strings in Java

- In Java, strings are objects of the class named `java.lang.String`.
- Example:

```
String s1= "Hello, World";
```



# Representing Strings in Java

- A string in Java is more abstract.
- That is, you aren't supposed to know about its internal structure, which makes it easy to use.
- Its methods allow a programmer to perform operations on it.

# Using the `String` Class

- The `String` class:
  - Is one of the many classes included in the Java class libraries.
  - Is part of `java.lang.package`.
  - Provides you with the ability to hold a sequence of characters of data.
- You will use the `String` class frequently throughout your programs.
- Therefore, it's important to understand some of the special characteristics of strings in Java.



# Topics

- Getting Started with Strings
- **String Class Documentation**
- String Class Methods
- Concatenating Strings
- Comparing Strings

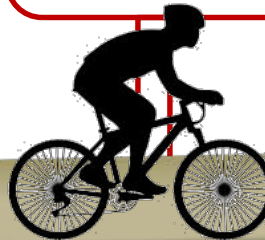
What is a  
Method?

The import  
Declaration  
and Packages

The String  
Class

The Random  
Class

The Math  
Class



Section 4

# Documentation of the `String` Class

You can access the documentation of the Java `String` class from here:

<https://docs.oracle.com/javase/8/docs/api/>

# Java Platform SE 8 Documentation for the String Class

Select All Classes  
or a particular  
package.

Details about the  
class selected

The classes for  
the selected  
packages are listed  
here.

The screenshot shows the Java Platform SE 8 Documentation for the String Class. The left sidebar, titled 'Java™ Platform Standard Ed. 8 DRAFT ea-b113', has a 'Packages' section where 'java.lang' is selected. Below it, a list of classes is shown, including 'String', 'StringBuffer', 'StringBuilder', 'System', 'Thread', 'ThreadGroup', 'ThreadLocal', 'Throwable', and 'Void'. The main content area, titled 'Class String', shows the class hierarchy: 'java.lang.Object' and 'java.lang.String'. It also lists 'All Implemented Interfaces: Serializable, CharSequence, Comparable<String>'. The class signature is 'public final class String extends Object implements Serializable, Comparable<String>, CharSequence'. A note at the bottom states: 'The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.'

# String Class Documentation: Method Summary

```
public int charAt(String str)
```

Return type of  
the method

Name of the  
method

Data type of the parameter that  
must be passed into the method

Method Summary	
Methods	
Modifier and Type	Method and Description
char	<b>charAt</b> (int index) Returns the char value at the specified index.
int	<b>codePointAt</b> (int index) Returns the character (Unicode code point) at the specified index.
int	<b>codePointBefore</b> (int index) Returns the character (Unicode code point) before the specified index.
int	<b>codePointCount</b> (int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	<b>compareTo</b> (String anotherString) Compares two strings lexicographically.
int	<b>compareToIgnoreCase</b> (String str) Compares two strings lexicographically, ignoring case differences.
String	<b>concat</b> (String str) Concatenates the specified string to the end of this string.

# String Class Documentation: Method Detail

Click here to get the detailed description of the method.

int	<b>indexOf</b> (String str)	Returns the index within this string of the first occurrence of the specified substring.
int	<b>indexOf</b> (String str, int fromIndex)	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Detailed description of the `indexOf()` method

Further details about parameters and return value are shown in the method list.

## indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value *k* for which:

```
this.startsWith(str, k)
```

If no such value of *k* exists, then -1 is returned.

### Parameters:

str - the substring to search for.

### Returns:

the index of the first occurrence of the specified substring, or -1 if there is no such occurrence.

# Topics

- Getting Started with Strings
- `String` Class Documentation
- `String` Class Methods
- Concatenating Strings
- Comparing Strings

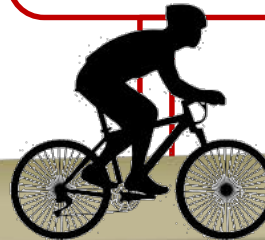
What is a  
Method?

The `import`  
Declaration  
and Packages

The `String`  
Class

The `Random`  
Class

The `Math`  
Class



Section 4

# String Methods: length

You can compute the length of a string by using the `length` method defined in the `String` class:

- Method: `name.length()`
- Returns the length, or the number of characters, in `name` as an integer value.
- Example:

```
String name= "Mike.W";  
System.out.println(name.length()); //6
```

# Accessing Each Character in a String

- You can access each character in a string by its numerical index.
- The first character of the string is at index 0, the next is at index 1, and so on.
- For example:

```
String str= "Hello, World";
```

H	e	l	l	o	,		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10	11

- `str` has 0 to 11 indexes; that is, between 0 to `str.length() - 1`.



# String Methods: `indexOf ( )`

- Each character of a string has an index.
- You can retrieve the index value of a character in the string by using the `indexOf` method:

Method	Description
<code>str.indexOf(char c)</code>	Returns the index value of the first occurrence of <code>c</code> in String <code>str</code> .
<code>s1.indexOf(char c,int beginIdx)</code>	Returns the index value of the first occurrence of <code>c</code> in String <code>str</code> , starting from <code>beginIdx</code> to the end of the string.

# String Methods: indexOf ( )

```
public static void main(String args[]){  
    String phoneNum = "404-543-2345";  
    int idx1 = phoneNum.indexOf('-');  
    System.out.println("index of first dash: "+ idx1);  
    //3  
    int idx2 = phoneNum.indexOf('-',4);  
    System.out.println("second dash idx: "+idx2); // 7  
}
```

# String Methods: charAt

- Returns the character of the string located at the index passed as the parameter.
- Method: `str.charAt(int index)`

```
String str="Susan";  
System.out.println(str.charAt(0)); //S  
System.out.println(str.charAt(3)); //a
```

# String Methods: substring( )

- You can extract a substring from a given string.
- Java provides two methods for this operation:

Method	Description
<code>str.substring(int beginIdx)</code>	Returns the substring from <code>beginIdx</code> to the end of the string.
<code>str.substring(int beginIdx, int endIdx)</code>	Returns the substring from <code>beginIdx</code> up to, but not including, <code>endIdx</code> .

# String Methods: substring( )

```
public static void main(String args[]){  
    String greeting = "Hello, World!";  
    String sub = greeting.substring(0, 5); → "Hello"  
    String w = greeting.substring(7, 11); → "Worl"  
    String tail = greeting.substring(7); → "World!"  
}
```

# String Methods: replace ( )

- This method replaces all occurrences of matching characters in a string.
- Method: `replace(char oldChar, char newChar)`
- Example:

```
public static void main(String args[]) {  
    String str= "Using String replace to replace character";  
    String newString =str.replace("r", "R");  
    System.out.println(newString);  
}
```

- Output: Using String Replace to Replace ChaRacter
- All occurrences of a lowercase “r” are replaced with a capital “R.”

# String Methods: `replaceFirst()`

- This method replaces only the first occurrence of a matching character pattern in a string.
- **Method:** `replaceFirst(String pattern, String replacement)`

# String Methods: replaceFirst ( )

- Example:

```
public static void main(String args[]) {  
    String replace = "String replace with replaceFirst";  
    String newString = replace.replaceFirst("re", "RE");  
    System.out.println(newString);  
}
```

- Output: String REplace with replaceFirst
- Only the first occurrence of "re" is replaced with "RE." The second occurrence isn't changed.





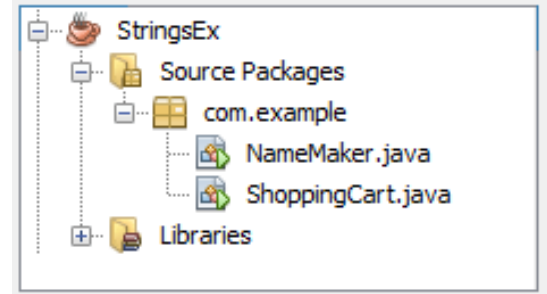
# Exercise 1, Part 1

- Import and open the `StringsEx` project.
- Examine `ShoppingCart.java`.
  - Perform the following:
    - Use the `indexOf` method to get the index for the space character (" ") within `custName`. Assign it to `spaceIdx`.
    - Use the `substring` method and `spaceIdx` to get the first name portion of `custName`. Assign it to `firstName` and print `firstName`.

## Exercise 1, Part 2



- You might notice that this project has two `.java` files with main methods.
  - This could seem like a contradiction because we said never to write more than one main method!
- Sometimes programmers do this when they're testing small bits of code and they want to keep all their files neatly in one project.
  - Unfortunately, pressing **run** in NetBeans always runs the same file and never the others.
  - You'll have to right-click the alternate file you want to run. A menu will appear with an option to run that file.



# Topics

- Getting Started with Strings
- `String` Class Documentation
- `String` Class Methods
- Concatenating Strings
- Comparing Strings

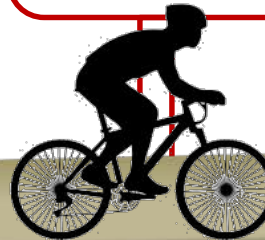
What is a  
Method?

The `import`  
IDeclaration  
and Packages

The `String`  
Class

The `Random`  
Class

The `Math`  
Class



Section 4

# Declaring and Creating a String

You can instantiate strings in two ways:

- String literals: Directly assign a string literal to a string reference.

String Reference

String Literal

```
String hisName = "Fred Smith";
```

- new operator: Similar to any other class. Not commonly used and not recommended.

```
String herName = new String("Anne Smith");
```

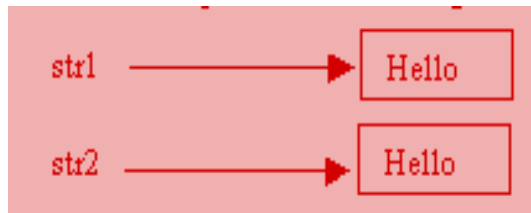
The new keyword

# Strings Are Immutable

- A `String` object is immutable; that is, after a `String` object is created, its value can't be changed.
- Because strings are immutable, Java can process them very efficiently.
  - Consider the following:

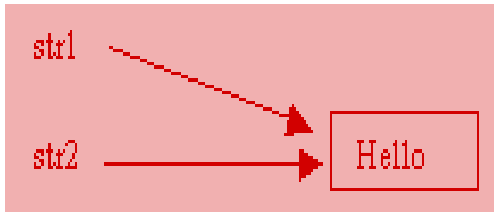
```
String str1 = "Hello";  
String str2 = "Hello";
```

- We expect this ...



# Strings Are Immutable

- But this is what happens ...



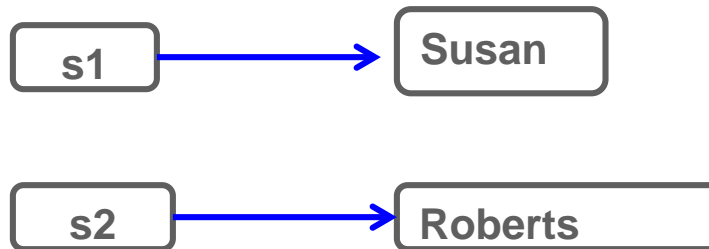
- The Java runtime system knows that the two strings are identical and allocates the same memory location for the two objects.

# Concatenating Strings

- In Java, string concatenation forms a new string that's the combination of multiple strings.
- You can concatenate strings in Java two ways :
  - + string concatenation operator
  - `concat ( )` method

# Using the + Operator (Before Concatenation)

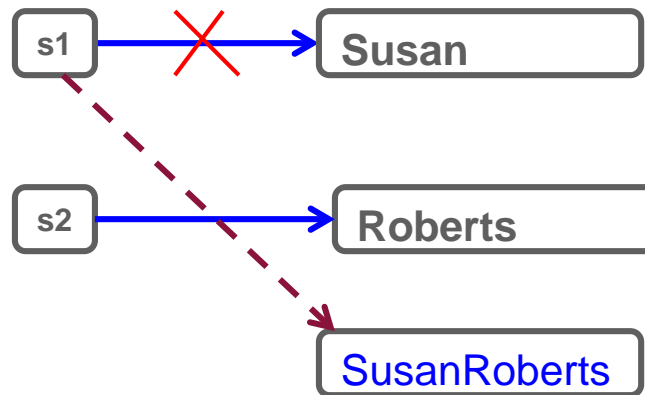
```
public static void main(String args[]) {  
    String s1 = "Susan";  
    String s2 = "Roberts";  
}
```





# Using the + Operator (After Concatenation)

```
public static void main(String args[]) {  
    String s1 = "Susan";  
    String s2 = "Roberts";  
    s1= s1+s2;  
    System.out.println(s1);  
}
```



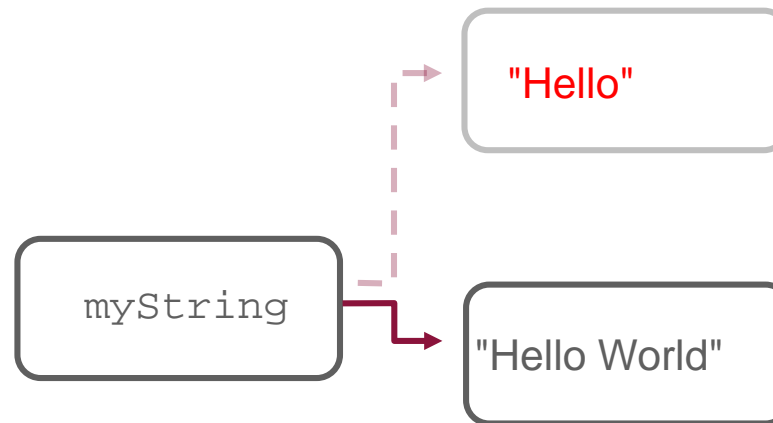
# Concatenating Non-String Data with String

- If one of the operands is a string, Java automatically converts non-string data types to strings prior to concatenation.
- Example:

```
public static void main(String args[]) {  
  
    String newString = "Learning Java"+8;  
    System.out.println(newString); // Learning Java 8  
  
    String numString = 8+8;  
    System.out.println(numString); //16  
  
    String numString1 = "8"+8;  
    System.out.println(newString1); // 88  
}
```

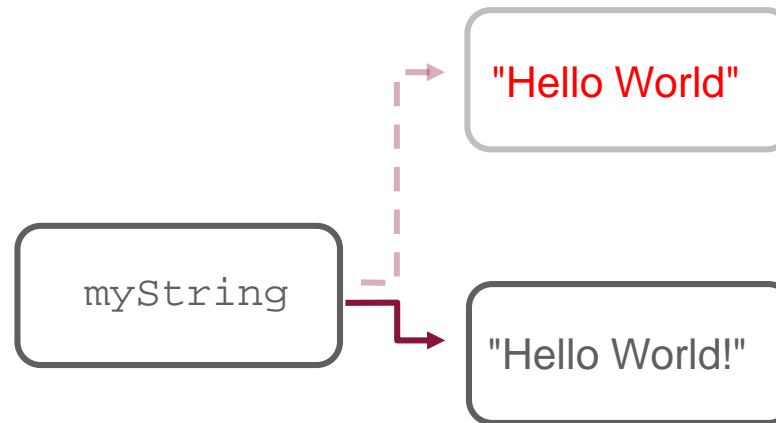
# Using the concat ( ) Method (Before Concatenation)

```
String myString = "Hello";  
myString = myString.concat(" World");
```



# Using the concat ( ) Method (After Concatenation)

```
String myString = "Hello";  
myString = myString.concat(" World");  
myString = myString + "!"
```





## Exercise 2

- Import and open the `StringsEx` project.
- Examine `NameMaker.java`.
- Perform the following:
  - Declare String variables: `firstName`, `middleName`, `lastName`, and `fullName`
  - Prompt users to enter their first, middle, and last names and read the names from the keyboard.
  - Set and display the `fullName` as `firstName`+a blank char+`middleName`+a blank char+`lastName`.



## Exercise 2

- Which do you think is preferable for this scenario?
- That is, the string concatenation operator or the `concat ( )` method?

# What's the Preferred Way to Concatenate Strings?

- As you observed in the previous exercise:
- `+` operator:
  - Can work between a string and a string, `char`, `int`, `double` or `float` data type value.
  - Converts the value to its string representation before concatenation.
- `concat ( )` method:
  - Can be called only on strings.
  - Checks for data type compatibility, and a compile time error is produced if they don't match.

# Topics

- Getting Started with Strings
- `String` Class Documentation
- `String` Class Methods
- Concatenating Strings
- Comparing Strings

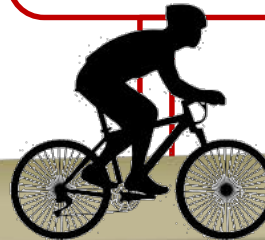
What is a  
Method?

The `import`  
Declaration  
and Packages

The `String`  
Class

The `Random`  
Class

The `Math`  
Class



Section 4



# How Do You Compare String Objects?

- You can compare two `String` objects by using the `compareTo` method.
- This method compares based on the lexicographical order of strings. Lexicographic comparisons are similar to the ordering found in a dictionary.
- The strings are compared character by character until their order is determined or until they prove to be identical.
- Syntax: `s1.compareTo(s2)`
- Returns an integer value that indicates the ordering of the two strings.



# Value Returned by `compareTo ( )`

- The integer value returned by the `compareTo ( )` method can be interpreted as follows:
  - Returns  $< 0$  when the string calling the method is lexicographically first.
  - Returns  $= 0$  when the two strings are lexicographically equivalent.
  - Returns  $> 0$  when the parameter passed to the method is lexicographically first.

# Using the compareTo Method

Let's look at some examples:

- `"computer".compareTo("comparison")`
  - Returns an integer  $> 0$  because the "comparison" parameter is lexicographically first.
- `"cab".compareTo("car")`
  - Returns an integer  $< 0$  because the "cab" string calling the method is lexicographically first.
- `"car".compareTo("car")`
  - Returns an integer equal to 0 because both are lexicographically equivalent.

# Using the compareTo method: Example

- Let's write a program to compare names by using the compareTo method:

```
public static void main(String[] args) {  
  
    String s1 = "Susan";  
    String s2 = "Susan";  
    String s3 = "Robert";  
  
    //Returns 0 because s1 is identical to s2  
    System.out.println(s1.compareTo(s2)); //Output is 0  
  
    //Returns >0 because 'S' follows 'R'  
    System.out.println(s1.compareTo(s3)); // Output is 1  
  
    //Returns <0 because 'R' precedes 'S'  
    System.out.println(s3.compareTo(s1)); // Output is -1  
}
```

# Summary

In this lesson, you should have learned how to:

- Locate the `String` class in the Java API documentation
- Understand the methods of the `String` class
- Compare two `String` objects lexicographically
- Find the location of a substring in a `String` object
- Extract a substring from a `String` object

