S t r i n g s

Objective #1: Use the String type and make use of its substring, indexOf, length and other methods.

There is no data type that can be used to create a primitive variable that can store a string (i.e. sequence of characters such as a word or phrase). Therefore, it is useful to use objects of the String type. String is a class that is available for use in any Java program without the necessity of having to include an import statement at the top of the program.



Unlike other objects, you don't have to use the new operator to instantiate a string object. The statement



String name = "John";

Declares the object name and initializes it with the string value "John" (not including the double quotation marks of course.)

It is legal to use the declaration statements

String name = new String();

String name = new String("John");

but you will rarely see a programmer do so.

Be wary though since the statement

String name;

Only declares the object variable name to be a reference to a String. name stores null and there is no corresponding instantiated String object. Thus the following code segment

String name;

System.out.println(name.length());

Causes a run­time null exception error since an actual object doesn't exist. The length method (which you learn in the following objectives) can't be executed.

However, the statement

String name = "";

Properly initializes name to the empty string and a String object is instantiated to the following code segment does execute

String name = "";

System.out.println(name.length());

And displays the value 0 since the length of the null string is zero.

The assignment statement,



1/7

name = "Sally";

changes the value of the object name (that has already been declared) to the value "Sally".

The concatenation operator + can be used to join two strings together. The following code segment



String firstName = "Jane";

String wholeName = firstName + " Smith"; System.out.println(wholeName);

causes the "Jane Smith" to be displayed.

To turn a number into a string, you can concatenate the empty string to an int or double



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int num = | 2; | = num + ""; // valid | |  |
| String word1 | |  |
| or use the static method valueOf from the String class as in... | | | |  |
| int num = | 2; | = String.valueOf(num); // valid | |  |
| String word1 | |  |
| but not |  |  |  |  |
| String word2 | | = num; | // error since you cannot directly assign |  |
| an int to | a String | |  |  |

If you use a plus symbol between an int or double and a string, Java will automatically convert the int or double into a string and concatenate everything together as in



int num = 2;

String word = "many"; System.out.println(num + word); // "2many"

Study the following example that purposefully concatenates null strings to int's:

int num1 = 20; int num2 = 13; int num3 = 8;

String mergedNumbers = "";

mergedNumbers = "" + num1 + "" + num2 + "" + num3; System.out.println(mergedNumbers); // "20138";

It is often useful to use the Integer class's parseInt method to turn the value of a string object, which happens to contain a number, into an integer value that can be assigned to an actual int variable. This process of turning a String object into a primitive int value is not considered to be casting. Casting only occurs when a programmer turns an int into a double or vice versa. The parseInt method is similar to the CInt method and the Val function in Visual Basic. In the following code segment,



2/7

String number = "12";

int num = Integer.parseInt(number); System.out.println(num + 3);

will cause cause the integer value 15 to be displayed. It is necessary to "parse" the value stored in the String object to convert it into a value that can be properly stored in an int primitive variable. Within the println method call, the sum of num plus 3 will simplify to 15 since mathematical addition occurs.

However the following code

String number = "12";

System.out.println(number + 3);

will cause the string output "123" to be displayed. Since number is a string object and not an integer primitive variable, Java interprets the plus symbol to be a string concatenation operator and not a mathematical addition operator. Even though the digit 3 is a numerical value it ends up being concatenated and not added to number. The rule is that if either one of the + operator's arguments are a String object, the + operator is treated as a concatenation operator. Only if the arguments on both sides of the plus symbol are integer or floating­point values will it be treated as a mathematical addition operator.

a neat aspect of the parseInt method is that it can be used to convert strings made up of binary digits into decimal numbers if the optional second parameter 2 is used. Since the binary number 1110 equals the decimal number 14, the variable num stores the number 14 after the following statement executes:

int num = Integer.parseInt("1110", 2);

In fact, this works for any numbering system including hexadecimal (base 16) and octal (base 8).

int num = Integer.parseInt("1F", 16); // num stores 31 int num = Integer.parseInt("77", 8); // num stores 63

Also, another neat thing that is that the toString method of the Integer class performs the opposite function of the parseInt method in this sense. The toString method will return the binary equivalent to a decimal number expressed as a string. For example, the string value "1110" will be stored in binaryNum after the following statement executes.

String binaryNum = Integer.toString(14, 2);

The parseDouble method of the Double class can be used to convert the value stored in a String object to a floating­point value and then store it into a double variable as in



String userPrice = "13.99";

double price = Double.parseDouble(userPrice);

The parseInt and parseDouble methods are static methods of the Integer and Double classes not instance methods like most that we've studied in this course so far. It is not necessary and not desirable to instantiate a Double object and calling the parseDouble method with that object. Rather, the name of the class Double is typed followed by the dot operator and then the name of the static method

3/7

parseDouble.

The substring method of the String class can be used to extract part of a string from a larger string value. The following code segment



String message = "abcdefghijklmnopqrstuvwxyz";

String firstPart = message.substring(5, 10);

System.out.println(firstPart);

would cause the whole message "fghij" to display. The substring method causes a string starting at the character in position 5 to be extracted. The first letter of a string is considered to be the zero position, the second letter is considered to be the 1 position, and so on. So, in this example, all the characters starting with the character 'f' up through BUT NOT INCLUDING the character in position 10 are extracted and stored into the string firstPart.

The substring method can also be called with only one integer argument. The following code segment

String message = "Four score and seven years ago";

String lastPart = message.substring(5);

System.out.println(lastPart);

would cause the phrase "score and seven years ago" to display since every character from position 5 to the end of the string is extracted from message and then stored in lastPart.

Note the following tricky examples



String word = "car"; System.out.println(word.substring(0)); // "car" System.out.println(word.substring(1)); // "ar" System.out.println(word.substring(2)); // "r"

System.out.println(word.substring(3)); // "" (empty string) System.out.println(word.substring(4)); // error

Java allows you to attempt to access the next position after the last character in a string. But it doesn't allow you to access a character that is two positions away from the last character in a string.

The length method of the String class can be used to obtain the length of a string object or string literal as counted by the number of characters (i.e. letters, digits, and symbols) that are stored there. The following code segment



String message = "Hello World"; int length = message.length(); System.out.println(length);

causes the value 11 to be displayed, since there are eleven characters including the blank space in the

String object message.

The length method can be used to determine the last letter of a string using this technique

4/7

System.out.println(name.substring(name.length() ­ 1));

Note that the length of the empty string is zero as in

String word = "";

System.out.println(word.length()); // displays 0

but do not mistakenly confuse "" with null. That is, you should realize that the null string "" (aka empty string) is not the same thing as null (i.e. the memory address 0).

String word;

System.out.println(word.length());

* causes null exception error since word was not instantiated
* and it is equal to null rather than the empty string

A set of characters (i.e. letters, digits, and symbols) placed inside of double quotes is called a string literal though it can also be considered a String object. Many String class methods can be used on a string literal.



|  |  |  |  |
| --- | --- | --- | --- |
| "Hello World".length(); | // | 11 since blank spaces count as | |
| characters and add to the length of a string | | | |
| "Hello World".substring(0, 5); // | | "Hello" | |
| "Hello World".substring(0, 6); // | | "Hello | " |
| "Hello World".substring(0, 7); // | | "Hello | W" |

The += operator can also be used to concatenate two strings. The code segment



String word1 = "Hello"; String word2 = "World"; word1 += word2;

displays the string word1 to be have the string value "HelloWorld". Note that the following two statements are equivalent:

word1 += word2;

word1 = word1 + word2;

The indexOf method is used to find a first occurrence of a substring (one or more characters) in a string. If the specified substring is not found in the string, the value ­1 is returned.



String word = "score";

String message = "Four score and seven years ago"; int position = message.indexOf(word); System.out.println(position);

causes 5 to be displayed since the s in "score" is in position 5 of the string message. Remember that the very first character in a string is considered to be position zero.

5/7

|  |  |  |
| --- | --- | --- |
|  | The equals method is used to compare two strings. The expression | |
|  | string1.equals(string2) | returns true if and only if string1 and string2 store the |
|  | same values. |  |



You will be studying the equals method in more detail later in this course.

The compareTo method is used to alphabetically compare two strings. The expression



string1.compareTo(string2)

returns a positive number if string1 is greater than string2. It returns the value zero if they are equal. It returns a negative number if string1 is less than string2.

You will be studying the compareTo method in more detail later in this course.

You can learn many other useful methods of the String class by viewing its  [Java AP](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html)I.



Objective #2: Understand that Strings are immutable.

No string method can change or modify the string object it is being called upon. In other words, none of the methods in the String class are modifiers. For this reason, strings are considered to be immutable. Most other classes that we will work with in this course are not immutable.



Review this code segment and it's comments:

String food1 = "bacon";

String food2 = "eggs";

food1 = food2; // food1 now stores "eggs"

food2 = "bagel"; // food2 now stores "bagel"

// but food1 still stores "eggs" since, unlike

other objects such as Bugs, Strings are immutable

|  |  |  |  |
| --- | --- | --- | --- |
| System.out.println(food1 + " | " + food2); // displays "eggs bagel" | |  |
| In the following code segment, |  |  |  |
| String name1 = "John Smith"; | // | name1 stores object reference A32 |  |
| & "John Smith" is stored in the | | object at A32 |  |
| String name2 = name1; | // | name2 stores A32 since A32 is |  |
| stored in name1 | // | name1 now stores object reference |  |
| name1 = name1.substring(5); |  |
| B52 and "Smith" is stored in | the object at B52 | |  |

* name2 still stores A32 and "John Smith" is still stored in the object at A32. name2 were aliased for a moment
* but the aliasing is broken with this last assignment statement

name1 becomes "Sm" but name2 is still "Smith". A new memory address (i.e. object reference) is given to name1 when the assignment statement executes. If strings were not immutable name2 would also be "Smith" and not "John Smith" at the end of the code segment.

6/7

Later in this course we will learn how aliasing works differently with regular objects such as Bug objects. Suppose you have a Bug object has a myColor property. In the following code segment

Bug flik = new Bug(Color.BLUE); Bug atta = flik; flik.setColor(20);

flik and atta both have the color of BLUE. In other words, what happens to flik's properties also happens to atta's properties since Bug objects are not immutable.

Here is another example, illustrating how immutable String objects are different from most other kinds of objects.



String name1 = "Fred"; String name2 = "Sally"; name2 = name1;

name1 += " Smith"; System.out.println(name1); // Fred Smith System.out.println(name2); // Fred

But consider this code segment with regular objects such as Bug objects:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bug flik1 | = | new Bug(Color.BLUE, 10); | // myDirection is 10 |  |
| Bug flik2 | = | new Bug(); | // myDirection is 0 |  |
| flik2 = flik1; | | |  |  |
| flik1.setColor(Color.RED); | | | // RED |  |
| System.out.println(flik1.getColor()); | | |  |

System.out.println(flik2.getWeight()); // BLUE

7/7