

## Unit 2: Using Objects

### Topic 6: Working with String Objects

Name: \_\_\_\_\_

Use your IntelliJ project from the Debugging Warm Up for this lab.

Here are three different ways to create `String` objects:

**1. Using a string literal:**

```
String str1 = "Thursday";
```

**2. Using the `String` class constructor with a string literal as the parameter:**

```
String str2 = new String("October");
```

**3. Using the `String` class constructor with another `String` object as the parameter :**

```
String s = "8th!";  
String str3 = new String(s);
```

We can the print out our `String` objects:

```
System.out.println(str1);
```

```
System.out.println(str2);
```

```
System.out.println(str3);
```

Thursday  
October  
8th!

**1. Free Style!** Create a new class (name it `LabRunner` or `Main` or whatever you want) and add a `main` method. Then write some code to create and print a few `String` objects using **all three** methods above.

Copy/paste your free style code:

**2.** Now, we've seen **String Concatenation** before, but let's get a little more nuanced. We can use **two** operators to **concatenate** (join) strings:

`+`            `+=`

**PREDICT** what this code segment will **print**:

```
int day = 8;  
String str1 = "Holy cow!";  
String str2 = "It's almost";  
String str3 = str1 + " " + str2;  
str3 += " ";  
str3 += "October " + day + ", ";  
str3 += 2022;
```

**My prediction:**

**Copy/paste and run the code segment above in your main method.** Were you *exactly* correct? If not, why not?

[Check](#)

**3. Now,** add a `println` statement at the end of the code above to print out `str3` and run the code segment again. *Now* did it match what you expected?

[See answer for real](#)

**Takeaway!** Primitive values (`int`, `double`, or `boolean`) can be **concatenated** (joined together) with `String` objects using `+` and `+=` operators. This causes “implicit conversion” of the primitive values to `String` objects so they can be joined together.

But the “rules” for how this is done are a little fuzzy.

**PREDICT** what the following will print:

```
String str1 = "A" + 1 + "B" + 2;  
String str2 = 1 + "A" + 2 + "B";  
String str3 = "A" + "B" + 1 + 2;  
String str4 = "A" + 1 + 2 + "B";  
String str5 = 1 + 2 + "A" + "B";  
String str6 = "A" + 1 + 2 + 3;  
String str7 = 1 + 2 + 3 + "A";  
String str8 = "1" + 2 + 3 + 4;  
String str9 = 1 + 2 + 3 + "4";  
String str10 = 3.5 + 2 + 3 + "4";  
System.out.println("str1 = " + str1);  
System.out.println("str2 = " + str2);  
System.out.println("str3 = " + str3);  
System.out.println("str4 = " + str4);  
System.out.println("str5 = " + str5);  
System.out.println("str6 = " + str6);  
System.out.println("str7 = " + str7);  
System.out.println("str8 = " + str8);  
System.out.println("str9 = " + str9);  
System.out.println("str10 = " + str10);
```

**PREDICTED OUTPUT:**

```
str1 =  
str2 =  
str3 =  
str4 =  
str5 =  
str6 =  
str7 =  
str8 =  
str9 =  
str10 =
```

**TEST!** Copy/paste the code above into your main method and run it to see if you were correct.

[Confirm](#)

Analyze the output (especially any that you predicted wrong), and come up with a “rule” for what happens when **primitives are concatenated with Strings using the `+` operator**:

**My rule:**

**Partner Share!** Discuss your rule with your partner -  
- did you both come up with the same rule for concatenating primitives with strings?

Partner's Name:

**Now try running this:**

```
String str11 = 1 + 2 + 3 + 4;  
System.out.println(str11);
```

**And then:**

```
String str12 = 45;  
System.out.println(str12);
```

**What happens? Why?**

[Here's one rule you might have discovered at this point](#)

**5. Let's do something similar, but with the += operator:**

**EXPLORE:** Copy/paste and run each set of code segments; what patterns do you notice in the printed values?

<pre>String str1 = "A"; str1 += 1; str1 += 2; str1 += 3; str1 += "B"; System.out.println("str1 = " + str1);  String str2 = "0"; str2 += 1; str2 += 2; str2 += 3; System.out.println("str2 = " + str2);  String str3 = "0" + 1; str3 += 2 + 3; System.out.println("str3 = " + str3);  String str4 = 1 + 2 + "3"; str4 += 4 + 5; str4 += 6; System.out.println("str4 = " + str4);</pre>	<pre>String str5 = "1" + 2 + 3; str5 += "4" + 5 + 6; System.out.println("str5 = " + str5);  String str6 = ""; str6 += 1; str6 += 2; str6 += 3 + 4 + 5; System.out.println("str6 = " + str6);  int five = 5; int six = 6; String s = "7"; s += 8; System.out.println(five + six + s);  String t = "2"; int f = 4; int x = 7; x += 5; System.out.println(t + f + x);</pre>
---	--

**Copy/paste printed output below for your notes:**

[check](#)

[check](#)

What “rules” can you come up with for using the += operator to concatenate?

Is it any different from the + operator?

**Takeaway!** When concatenating primitive values with `String` objects using `+` and `+=` operators it generally works as expected; *however...*

With the `+` operator, if there is a *mathematical* expression that appears at the *beginning* of a `String` assignment that uses concatenation, the *mathematical* expression evaluates first, *before* it is concatenated to a `String`.

Examples:

`String str5 = 1 + 2 + "A" + "B";` → **str5 is now 3AB (a String)**

**or**

`String str9 = 1 + 2 + 3 + "4";` → **str9 is now 64 (a String "64")**

Similarly, with the `+=` operator, if a *standalone* mathematical expression is being appended, the mathematical expression evaluates first:

Example:

`str3 += 2 + 3;` → **this is the same as `str3 += 5;`**

**LAB CONTINUES ON NEXT PAGE**

6. Try printing the following message to your console *exactly* as shown (including quotes) using a print or println statement:

```
My cat says "meow" when she is happy!
```

So it appears like this:

```
CONSOLE SHELL
My cat says "meow" when she is happy!
```

What error do you get? Can you figure out what the problem is?

There are a few characters that you can't *directly* print in Java -- **quotation marks** are one!

Trying to print quote marks inside of a string will cause an issue:

```
System.out.println("My cat says "meow" when she is happy!");
```

Java thinks the " before meow is *closing* the opening quote around the string:

```
System.out.println("
```

This confuses Java, which then spews out multiple unintelligible errors:

```
Main.java:5: error: ')' expected
    System.out.println("My cat says "meow" when she is happy!");
                                ^
Main.java:5: error: not a statement
    System.out.println("My cat says "meow" when she is happy!");
                                ^
Main.java:5: error: ';' expected
    System.out.println("My cat says "meow" when she is happy!");
                                ^
```

7. Copy/paste/run this line in Replit and see what character causes the error:

```
System.out.println("This is a backslash \ and this is a forward slash /");
```

You will get the *same* error if you try to save it to a String:

```
String str = "This is a backslash \ and this is a forward slash /";
```

Which character causes the issue?

[confirm](#)

If you said “the backslash!” you were correct ☐ But looking at the error...

```
Main.java:5: error: illegal escape character
    System.out.println("This is a backslash \ and this is a forward slash /");
                                   ^
```

...what’s an **escape character**?

If you want to print a quote mark or a backslash, or include it in a `String`, you must use something called an **escape sequence**, which is a `\` and an “escape character”:

**Here are the three escape sequences to know and use!**

To print this character...	...use this <b>escape sequence</b>
"	<code>\ "</code>
<code>\</code>	<code>\\</code>
new line	<code>\n</code>

### EXAMPLES!

**EXAMPLE 1: To print this:**

My cat says "meow" when she is happy!

**Use the `\"` escape sequence:**

```
System.out.println("My cat says \"meow\" when she is happy!");
```

**EXAMPLE 2: To print this:**

This is a backslash `\` and this is a forward slash `/`

**Use the `\\` escape sequence:**

```
System.out.println("This is a backslash \\ and this is a forward slash /");
```

**THIS ONE IS COOL!**

**EXAMPLE 3: To print *multiple lines* with a *single print statement*:**

CS is  
the  
BEST!

**Use the `\n` (new line) escape sequence:**

```
System.out.println("CS is\nthe\nBEST!"); //note: no spaces!
```

**EXAMPLE 4: And this line of code:**

```
System.out.println("a backslash \\ and a quote \" and look!\na new line!");
```

**Prints this:**

```
a backslash \ and a quote " and look!  
a new line!
```

**EXAMPLE 5: These two lines of code:**

```
String str = "The best punctuation\nmark is: \"\\\"\\nSay \"yes\\\"!";  
System.out.println(str);
```

**Prints this:**

```
The best punctuation  
mark is: "\"  
Say "yes"!
```

8. Use escape characters to print each of the following with a **SINGLE print or println statement**; **figure it out first, then test in Replit!** Copy/paste the single print statement you used to print each string. *One is done as an example.*

**Example:**

```
So just what is an  
"escape character" exactly?
```

**Copy/paste your single print statement below:**

```
System.out.println("So just what is an\n\"escape character\" exactly?");
```

**A.** Cows say "moo"!

**B.** Brooklyn Tech  
29 Ft. Greene  
Brooklyn

**C.** An "i"  
for  
an eye

**D.** The string ""  
is an empty string.

**E.** The escape sequence for \ is \\

**F.** The escape sequence for " is \"

**G.** /A\V/A\  
\V/A\V/

**H.** Use "\n"  
for a new line.

**I.** My favorite punctuation mark is "\"

**J.** Now I know my  
A  
B  
C's

**K.** "\\_("/)\\_/"

**L.** Dear Jen,  
You got an "A"!  
  
Sincerely,  
The teacher

**M.** \* \*  
\* \* \*  
\* \*




**CREATE YOUR OWN!** On the right, come up with your *own* statement or image that uses *at least* one quotation mark ("), backslash (\), *and* a new line.

**Now figure out how to print your creation with a single print statement :p**  
**Copy/paste your print statement below:**

(optional) A challenge! ☐ Are you up to it?!

L.



A diagram of a hexagon with internal lines forming a smaller hexagon and connecting vertices to midpoints of opposite sides. The vertices of the outer hexagon are labeled with single quotes (') and the vertices of the inner hexagon are labeled with double quotes (").

**10.** Consider the following code segment.

**Determine (*using just your brains!*) what the output will be:**

```
String name = "Angie";
int age = 17;
String word = "Bam";
String info = "Name:\n" + name + "\n" + "\nAge:\n" + age;
info += "\n\nFavorite Word:\n" + "\"" + word + "!\"";
System.out.println(info);
```

**Prediction: This code will print:**

**TEST!** *Confirm your answer above by copying/pasting into your main method and running it.*

*confirm*

This syntax:

+ "\n" +

Seems a little strange; what does it do exactly?

*Confirm*

**11. Consider the following code segment. What does it print? (brains only at first!)**

**Note the use of `print` instead of `println`!**

```
String greeting = "Hello, friend!\n";
```

```
System.out.print(greeting);  
System.out.print("Nice to see you!\n");  
System.out.println("Goodbye!");
```

**Prediction: This code will print:**

**TEST!** *Confirm* your answer above by copying/pasting into Replit and running it!

[confirm](#)

**What relationship do you notice between something like:**

```
System.out.print("Nice to see you!\n");  
System.out.println("Goodbye!");
```

**And**

```
System.out.println("Nice to see you!");  
System.out.println("Goodbye!");
```

**What do you notice?** (if you aren't sure, run both code segments to see!)

**LAB CONTINUES ON NEXT PAGE**

12. Consider the following BabyParrot class.

**Note how the name instance variable is set to a default value of the *empty string*.**

```
public class BabyParrot
{
    private String species;
    private int age;
    private String name;

    public BabyParrot(String species, int age)
    {
        this.species = species;
        this.age = age;
        this.name = ""; // use a default value of the 'empty string'
    }

    public void setName(String newName)
    {
        name = newName;
    }

    // note this method RETURNS a String
    public String parrotInfo()
    {
        String str = "Species: " + species + "\n";
        str += "Name: " + name + "\n";
        str += "Age: " + age;
        return str;
    }
}
```

The following **code segment** appears in a class other than BabyParrot (such as a runner class main method):

```
BabyParrot baby = new BabyParrot("Toucan", 3);
baby.setName("Lil' Beans");
String info = baby.parrotInfo(); // store returned string in variable
System.out.println(info);        // print the returned string
```

**Determine (using just your brains) what the output will be:**

<b>This code will print:</b>	
------------------------------	--

**TEST!** Confirm your answer above by copying/pasting into Replit and running it!

You will need to **copy/paste the BabyParrot class** into a new file named **BabyParrot.java**, and copy paste the code segment shown above into a client/runner class `main` method.

[Confirm output](#)

**13.** Add a new instance variable to the `BabyParrot` class of type `double` called `weight`, and initialize it in the constructor from a parameter (so add it as a parameter to the constructor).

**14.** Then, **modify** the `parrotInfo` method to include the `weight` in the returned string such that this code (with a weight of **4.25** added):

```
BabyParrot baby = new BabyParrot("Toucan", 3, 4.25);
baby.setName("Lil' Beans");
String info = baby.parrotInfo();
System.out.println(info);
```

**Prints this:**

```
Species: Toucan
Name: Lil' Beans
Age: 3
Weight: 4.25
```

**Copy/paste your updated BabyParrot class below:**

[Compare](#)

**Lab continues on the next page**

**15.** Below is the Student class from the sample Student Program; copy/paste it and add it to your project.

```
public class Student {
    /* Instance Variables */
    private String firstName;
    private String lastName;
    private int gradYear;
    private double accumulatedTestScores;
    private int testScoreCount;

    /* Constructor */
    public Student(String firstName, String lastName, int gradYear) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.gradYear = gradYear;
        accumulatedTestScores = 0.0;
        testScoreCount = 0;
    }

    /* Getter Methods */
    // returns firstName
    public String getFirstName() {
        return firstName;
    }

    // returns lastName
    public String getLastName() {
        return lastName;
    }

    /* Setter Methods */
    // Sets gradYear to newGradYear
    public void setGradYear(int newGradYear) {
        gradYear = newGradYear;
    }

    // Adds newTestScore to accumulatedTestScores
    // and increments testScoreCount by 1
    public void addTestScore(double newTestScore) {
        accumulatedTestScores += newTestScore;
        testScoreCount++;
    }

    // Returns true if the student's average test score is greater
    // than or equal to 65; returns false otherwise (see Note 2 below)
    public boolean isPassing() {
        if (averageTestScore() >= 65) {
            return true;
        } else {
            return false;
        }
    }

    // Returns the Student's average test score as the
```

```
// quotient of accumulatedTestScores and testScoreCount
public double averageTestScore() {
    double average = accumulatedTestScores / testScoreCount;
    return average;
}

// this method prints all info of a Student object to the console
public void printStudentInfo() {
    System.out.println("Student Full Name: " + firstName + " " + lastName);
    System.out.println("Graduation Year: " + gradYear);
    System.out.println("Number of tests: " + testScoreCount);
    System.out.println("Average Test Score: " + averageTestScore());
    System.out.println("Is passing: " + isPassing());
}

// ADD YOUR NEW getStudentInfo() METHOD HERE

}
```

There is already a `printStudentInfo()` method that prints out the information about the Student object. **Add** a new method, `getStudentInfo()`, that *returns* a String constraining the same information, in the same order, as the information that is printed by the `printStudentInfo()` method. The String object returned by your new method, when printed, should produce the same output as the `printStudentInfo()` method.

#### TEST CODE:

```
Student student = new Student("Abby", "Smith", 2023);
student.addTestScore(95);
student.addTestScore(98);
System.out.println(student.getStudentInfo());
```

#### EXPECTED OUTPUT:

```
Student Full Name: Abby Smith
Graduation Year: 2023
Number of tests: 2
Average Test Score: 96.5
Is passing: true
```

Copy/paste your `getStudentInfo` method below:

[Compare](#)

**Done!**

Submit in Google Classroom:

Turn in

Answer ([back](#))

**It prints NOTHING!** ☐ There is no `print` or `println` statement to actually print `str3`! ☐



Real Answer ([back](#))

Adding this: `System.out.println(str3)` prints the following:

```
Holy cow! It's almost October 8, 2022
```

Answer ([back](#))

**One rule you might have discovered this point:**

When concatenating primitives with `Strings` with the `+` operator, if a mathematical expression (i.e. more than two numbers with an operation in between) appears at the ***beginning*** of the expression, such as:

```
String str9 = 1 + 2 + 3 + "A";  
System.out.println(str9);
```

Which prints: **6A**

...the *mathematical* expression  $(1 + 2 + 3)$  evaluates *first*, and then the result is converted to a `String` and concatenated.

However, if a mathematical expression appears in the *middle* or *end*, Java converts each primitive to a `String` *individually* and then concatenates all the individual `Strings`:

```
String str6 = "A" + 1 + 2 + 3;  
System.out.println(str6);
```

Which prints: **A123**

Lastly, as you saw these two lines, *both* which cause a compiler error:

```
String str11 = 1 + 2 + 3 + 4;  
  
String str12 = 12;
```

The expression you are assigning to a `String` must contain *at least one* `String` component, otherwise Java will ding you for using the wrong type! Above, `str11` is being assigned the sum of four `ints` (none are strings), and `str12` is being assigned an `int`.

Confirm ([back](#))

Note in particular **str6** vs. **str7**, and **str8** vs. **str9**

```
str1 = A1B2
str2 = 1A2B
str3 = AB12
str4 = A12B
str5 = 3AB
str6 = A123
str7 = 6A
str8 = 1234
str9 = 64
str10 = 8.54
```

## Output ([back](#))

```
String str1 = "A";
str1 += 1;
str1 += 2;
str1 += 3;
str1 += "B";
System.out.println("str1 = " + str1);

String str2 = "0";
str2 += 1;
str2 += 2;
str2 += 3;
System.out.println("str2 = " + str2);

String str3 = "0" + 1;
str3 += 2 + 3;
System.out.println("str3 = " + str3);

String str4 = 1 + 2 + "3";
str4 += 4 + 5;
str4 += 6;
System.out.println("str4 = " + str4);
```

### PRINTED OUTPUT

```
str1 = A123B
str2 = 0123
str3 = 015
str4 = 3396
```

```
String str5 = "1" + 2 + 3;
str5 += "4" + 5 + 6;
System.out.println("str5 = " + str5);

String str6 = "";
str6 += 1;
str6 += 2;
str6 += 3 + 4 + 5;
System.out.println("str6 = " + str6);

int five = 5;
int six = 6;
String s = "7";
s += 8;
System.out.println(five + six + s);

String t = "2";
int f = 4;
int x = 7;
x += 5;
System.out.println(t + f + x);
```

### PRINTED OUTPUT

```
str5 = 123456
str6 = 1212
1178
2412
```

Confirm ([back](#))

**THIS:**

```
System.out.println("This is a backslash \ and this is a forward slash /");
```

**LEADS TO THIS:**

```
Main.java:5: error: illegal escape character
    System.out.println("This is a backslash \ and this is a forward slash /");
                                   ^
```

If you notice, the ^ symbol is pointing to the space after the **backslash**! The backslash, \, is another symbol that can't be directly printed, like quote marks.

Confirm ([back](#))

CONSOLE

SHELL

```
Species: Toucan  
Name: Lil' Beans  
Age: 3
```

Compare ([back](#))

Necessary changes are in orange:

```
1  public class BabyParrot
2  {
3      private String species;
4      private int age;
5      private String name;
6      private double weight;
7
8      public BabyParrot(String species, int age, double weight)
9      {
10         this.species = species;
11         this.age = age;
12         this.name = ""; // use a default value of the 'empty string'
13         this.weight = weight;
14     }
15
16     public void setName(String name)
17     {
18         this.name = name;
19     }
20
21     public String parrotInfo()
22     {
23         String str = "Species: " + species + "\n";
24         str += "Name: " + name + "\n";
25         str += "Age: " + age + "\n";
26         str += "Weight: " + weight;
27         return str;
28     }
29 }
```

Confirm ([back](#))

You should see this; note the **blank lines** in between!

```
Name:
Angie

Age:
17

Favorite Word:
"Bam!"
```



Confirm ([back](#))

As shown in the output below, the `+ "\n" +` adds an **extra blank line** in between where the name is printed and where “Age:” is printed:

```
Name:
Angie

Age:
17

Favorite Word:
"Bam!"
```

Confirm ([back](#))

```
Hello, friend!  
Nice to see you!  
Goodbye!
```

Compare ([back](#))

Here is one way to do it, appending a  `+ "\n"` at the *end* of each part of the string:

```
public String getStudentInfo() {  
    String info = "Student Full Name: " + firstName + " " + lastName + "\n";  
    info += "Graduation Year: " + gradYear + "\n";  
    info += "Number of tests: " + testScoreCount + "\n";  
    info += "Average Test Score: " + averageTestScore() + "\n";  
    info += "Is passing: " + isPassing();  
    return info;  
}
```

Here is another way to do it, inserting a  `"\n"` at the *start* of each part of the string:

```
public String getStudentInfo() {  
    String info = "Student Full Name: " + firstName + " " + lastName;  
    info += "\nGraduation Year: " + gradYear;  
    info += "\nNumber of tests: " + testScoreCount;  
    info += "\nAverage Test Score: " + averageTestScore();  
    info += "\nIs passing: " + isPassing();  
    return info;  
}
```