ORACLE
Academy

Java Fundamentals

4-4

Strings

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Instantiate (create) a String
  - Describe what happens when a String is modified
  - Use the + and += operators for concatenating Strings
  - Interpret escape sequences in String literals

Overview

- This lesson covers the following topics:
  - Recognize the difference between a String and a primitive char data type
  - Test Strings with the compareTo() and equals() method
  - Describe why the == operator does not always work when testing String equality
  - Use String methods length(), substring(), indexOf(), and charAt()

What is a String?

- A String is an object that contains a sequence of characters
- Declaring and instantiating a String is much like any other object variable
- However, there are differences:
  - They can be instantiated (created) without using the new keyword
  - They are immutable
    - Once instantiated, they are final and cannot be changed

ORACLE Academy

Modifying a String

- Attempting to modify a String does not modify it, it creates a new String object
- As a new programmer, you will not notice this difference
- However, it becomes important in real systems where processing time is a key element in program design

String Operations Example

```
public class StringOperations{
   public static void main(String[] args){
      String string1 = "Hello";
      String string2="Lisa";
      String string3="";           //empty String or null
      string3="How are you ".concat(string2);
      System.out.println("string3: "+ string3);
      //get length
      System.out.println("Length: "+ string1.length();
      //get substring beginning with character 0, up to, but
      //not including character 5
      System.out.println("Sub: "+ string3.substring(0,5));
      //uppercase
      System.out.println("Upper: "+string3.toUpperCase());
   }//end method main
}//end class StringOperations
```

ORACLE
Academy

Class Template

- Use the following class to insert examples from this section

```
import java.util.Scanner;

public class StringPractice{
   public static void main(String[] args){
      //paste practice code here
   }//end method main

}//end class StringPractice
```

Instantiating a String

- Strings are object reference types
- They can be instantiated in two ways:
  - The new operator
  - String literals
  - There is no difference between the Strings below
  - Both methods of instantiation create identical objects

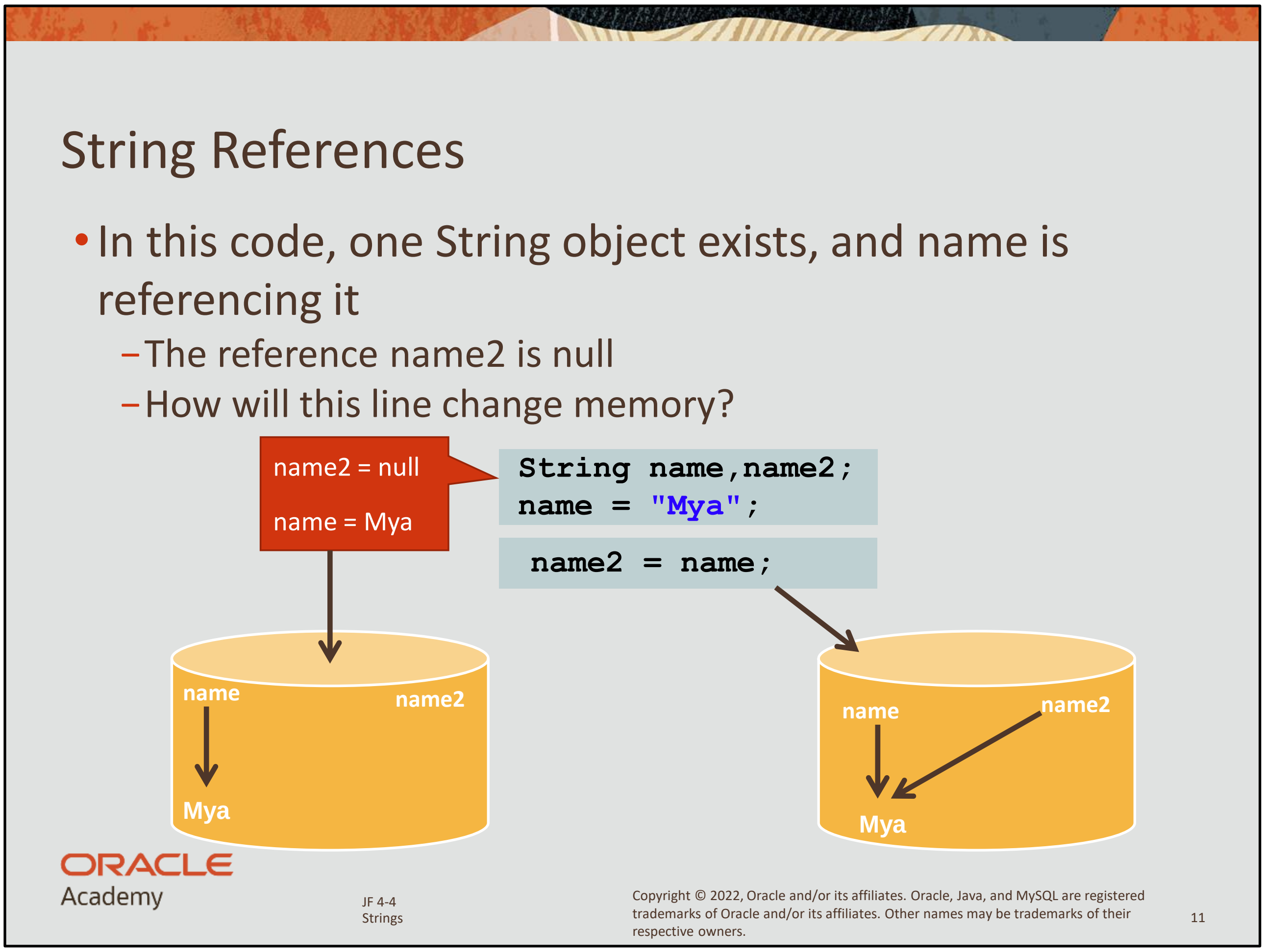
```
String s1 = new String("abc");  // new operator
String s2 = "abc";              // String literals
```

String References

- When you create a reference to an Object or String, the object does not necessarily exist yet
- In the code below, since name is not initialized, the program will not compile
- The variable name is a null pointer

```
String name;
System.out.println("My name is " + name);
```

Name = null



String References

- In this code, one String object exists, and name is referencing it
  - The reference name2 is null
  - How will this line change memory?

```
String name,name2;
name = "Mya";
```

```
name2 = name;
```

name2 = null
name = Mya
name
name2
Mya
name
name2
Mya

ORACLE Academy

JF 4-4
Strings

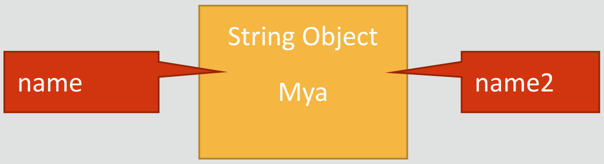
A reference actually stores the address of where the object is in memory. name and name2 both have the same address stored and thus point to the same object, "Mya", in memory.

String References

- Executing this line of code changes the name2 reference

```
name2 = name;
```

- The reference name2 will now refer to the same object as name

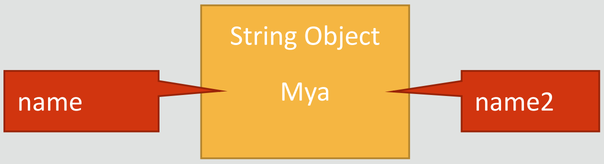


String References

- Executing these lines of code creates two instances of the String Mya in memory

```
String name, name2;
name = "Mya";
name2 = "Mya";
```

- Here the JVM compiler decides to save space and stores only one String Object that holds the String Mya

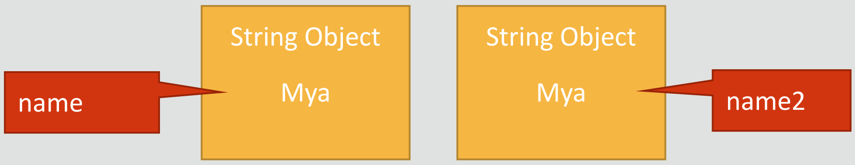


String References

- If we run a very similar program but ask the user to enter Strings, we get a different result

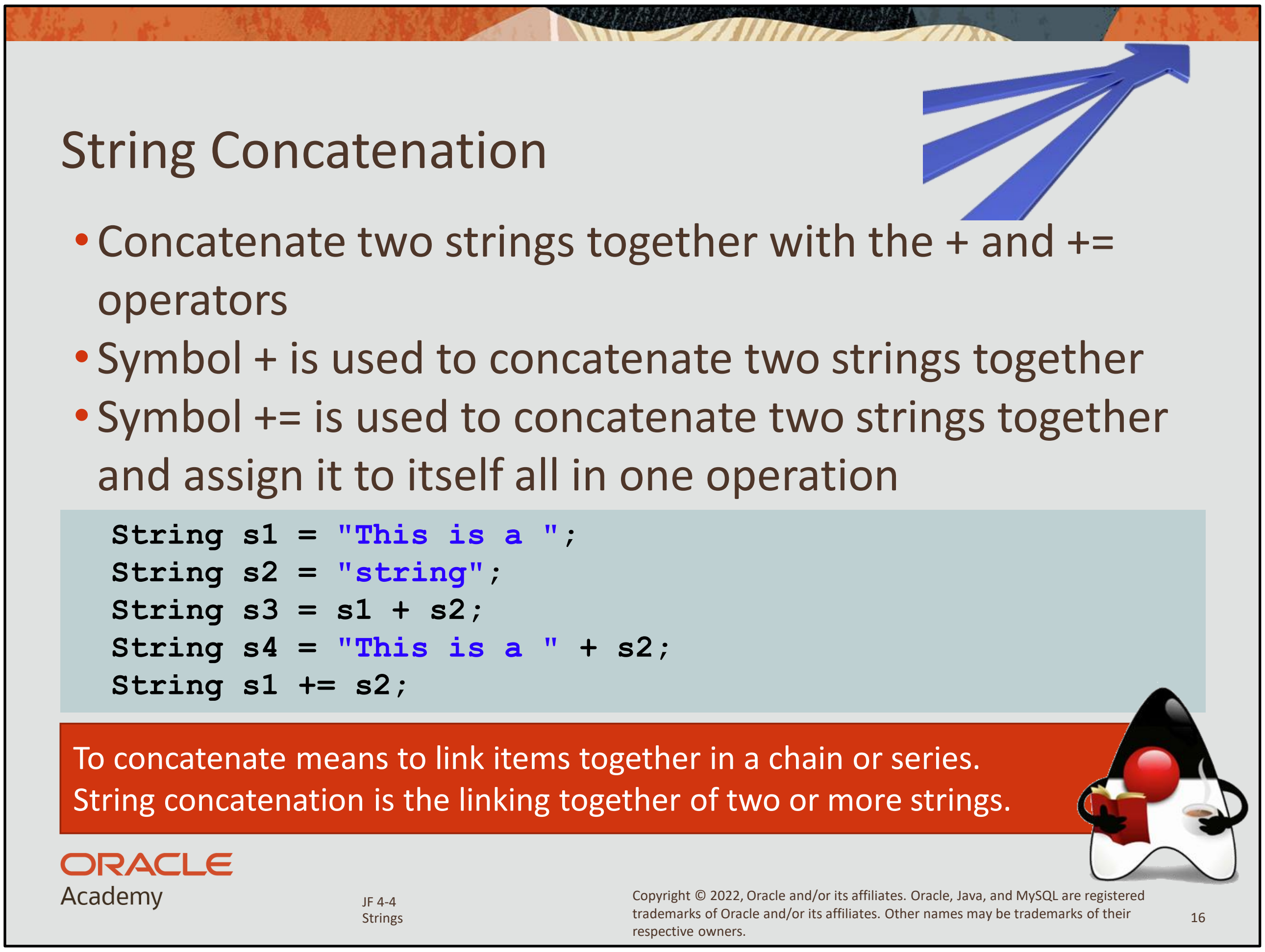
```
Scanner in= new Scanner(System.in);
String name, name2;
name=in.next();
name2=in.next();
```

- If the user types Mya for both Strings, the compiler actually creates two different String Objects



String References

- It is very difficult to predict when Java will create a new String or use an existing object for a String that is identical to the new String



String Concatenation

- Concatenate two strings together with the + and += operators
- Symbol + is used to concatenate two strings together
- Symbol += is used to concatenate two strings together and assign it to itself all in one operation

```
String s1 = "This is a ";
String s2 = "string";
String s3 = s1 + s2;
String s4 = "This is a " + s2;
String s1 += s2;
```

To concatenate means to link items together in a chain or series.
String concatenation is the linking together of two or more strings.

ORACLE Academy

JF 4-4
Strings

Experiment with the += operator on an integer type variable (X += 10;). Also practice using the -=, *=, /=, and even %= operators to see what results they return with integer values.

Manipulation of Strings

- Each time a String is changed, a new String is created in a new memory location and the reference is moved to the new location
- Some Strings will point to the same reference object in memory
- Making a modification to one String that is pointing to the shared reference will not make a modification to the others
- The one that you modify points to the new memory address that holds the new/modified string and the others still point to the original one

String Concatenation Example

- What will be the output of s1, s2, s3, and s4 at the end of these statements?

```
String s1 = "This is a ";
String s2 = "string";
String s3 = s1 + s2;
String s4 = "This is a " + s2;
String s1 += s2;
System.out.println("s1: " + s1);
System.out.println("s2: " + s2);
System.out.println("s3: " + s3);
System.out.println("s4: " + s4);
```

- Enter these statements in a Java program and check the results
- Did you guess correctly?

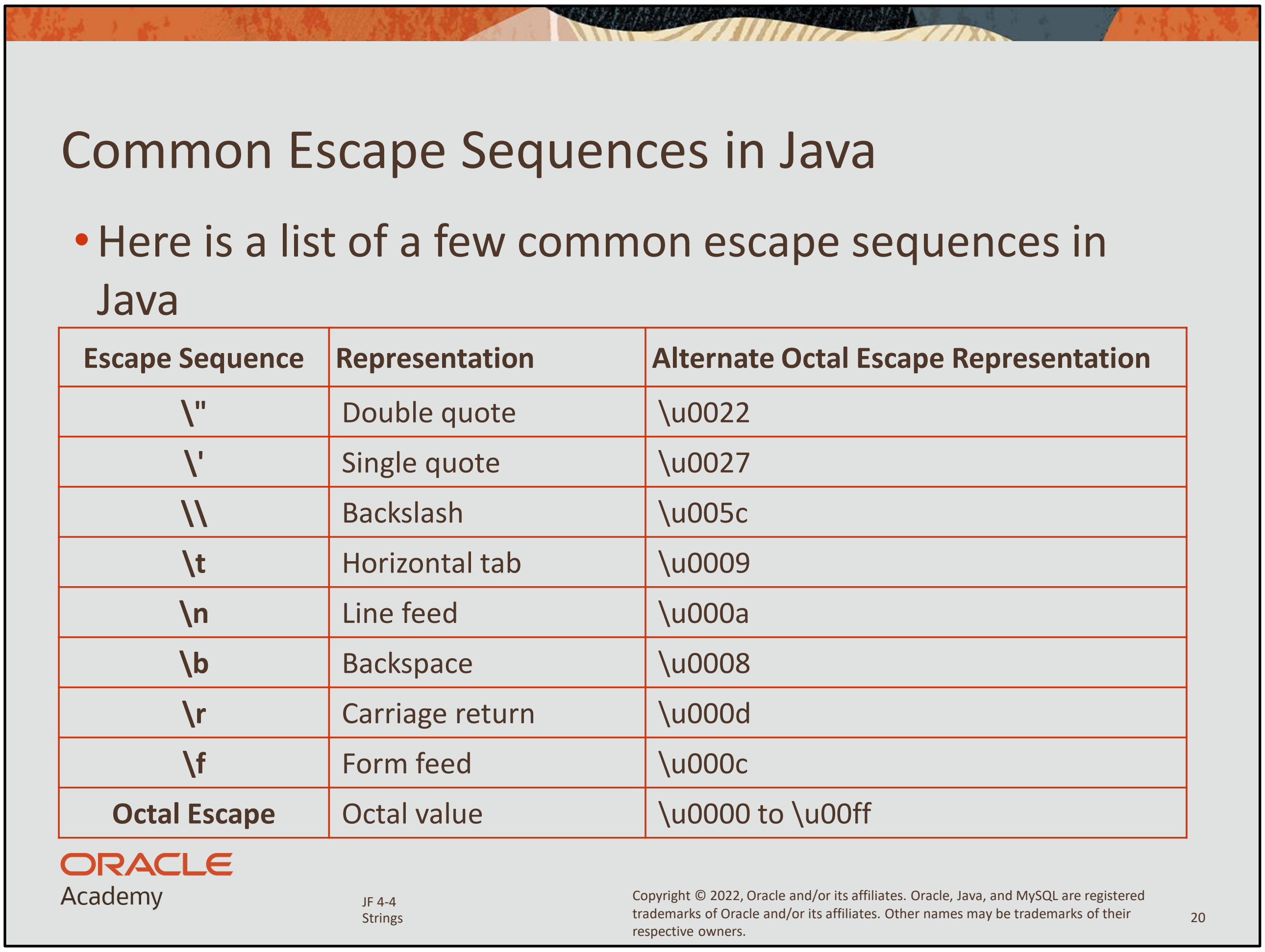
ORACLE
Academy

Escape Sequences \

- Escape sequences in string literals allow the user to add characters that would be misinterpreted by the compiler
- For example, to include a double-quote in your String, the compiler would interpret the double-quote as the beginning or ending of your String rather than including it
- Use the escape sequence \" to include a double-quote

```
String s1 = "This is an example of an escape.
        \n And now we're on a new line. \n \t This is a tab."
String s2 = "\"This is a quote \"";
System.out.println(s1);
System.out.println(s2);
```

ORACLE
Academy



Common Escape Sequences in Java

- Here is a list of a few common escape sequences in Java

Escape Sequence	Representation	Alternate Octal Escape Representation
\"	Double quote	\u0022
\'	Single quote	\u0027
\	Backslash	\u005c
\t	Horizontal tab	\u0009
\n	Line feed	\u000a
\b	Backspace	\u0008
\r	Carriage return	\u000d
\f	Form feed	\u000c
Octal Escape	Octal value	\u0000 to \u00ff

ORACLE
Academy

compareTo() Method

- Methods to use when comparing Strings
  - Method: s1.compareTo(s2)
  - Should be used when trying to find the lexicographical order of two strings
  - Returns an integer
  - If s1 is less than s2, an int < 0 is returned
  - If s1 is equal to s2, 0 is returned
  - If s1 is larger than s2, an int > 0 is returned

The compareTo() method is needed in sorting algorithms.

equals() Method

- Methods to use when comparing Strings
  - Method: s1.equals(s2)
  - Should be used when you only wish to find if the two strings are equal
  - Returns a boolean value
  - If true is returned, s1 is equal to s2
  - If false is returned, s1 is not equal to s2

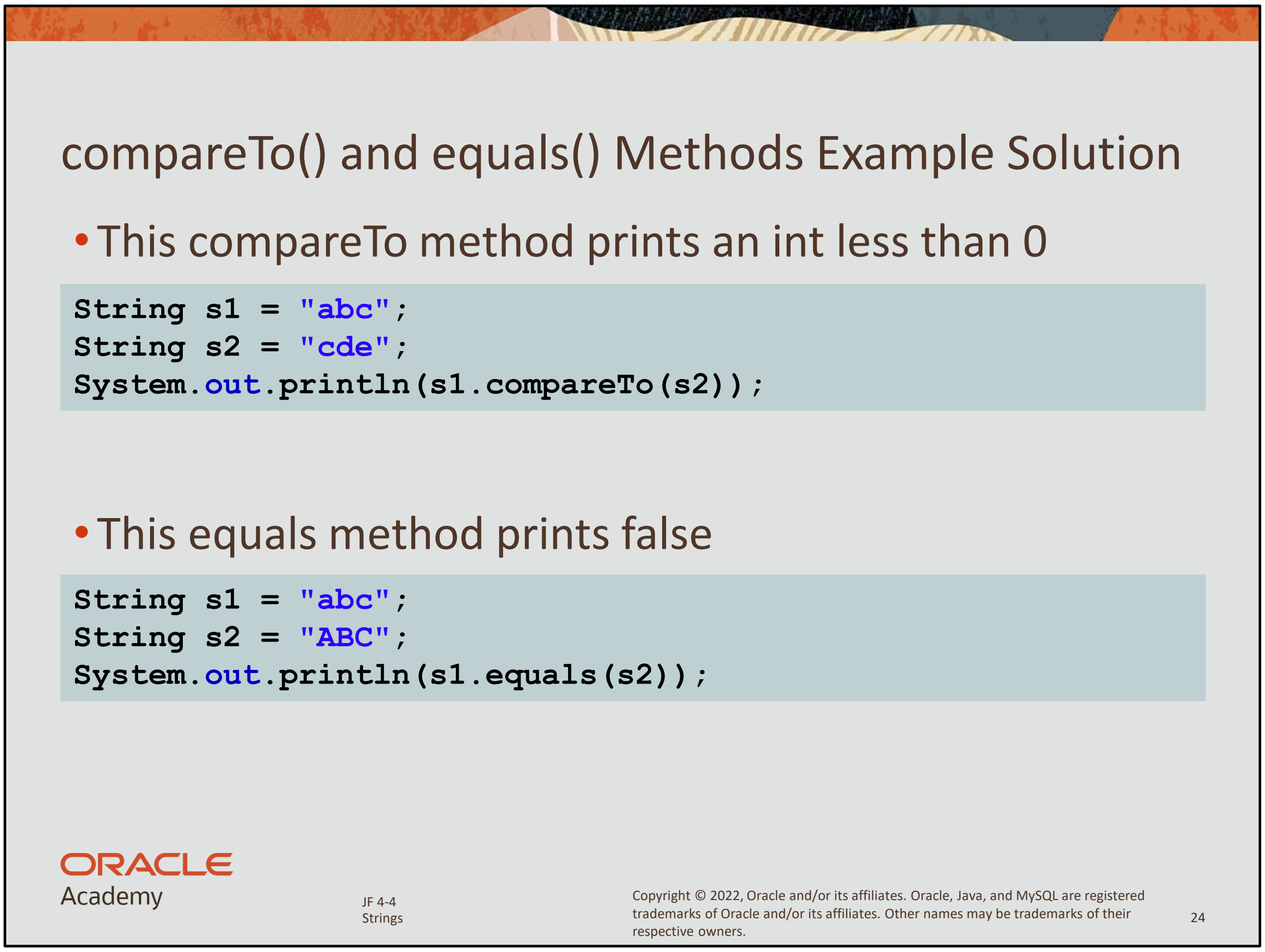
compareTo() and equals() Methods Example

- What will this compareTo() method print?

```
String s1 = "abc";
String s2 = "cde";
System.out.println(s1.compareTo(s2));
```

- What will this equals() method print?

```
String s1 = "abc";
String s2 = "ABC";
System.out.println(s1.equals(s2));
```



compareTo() and equals() Methods Example Solution

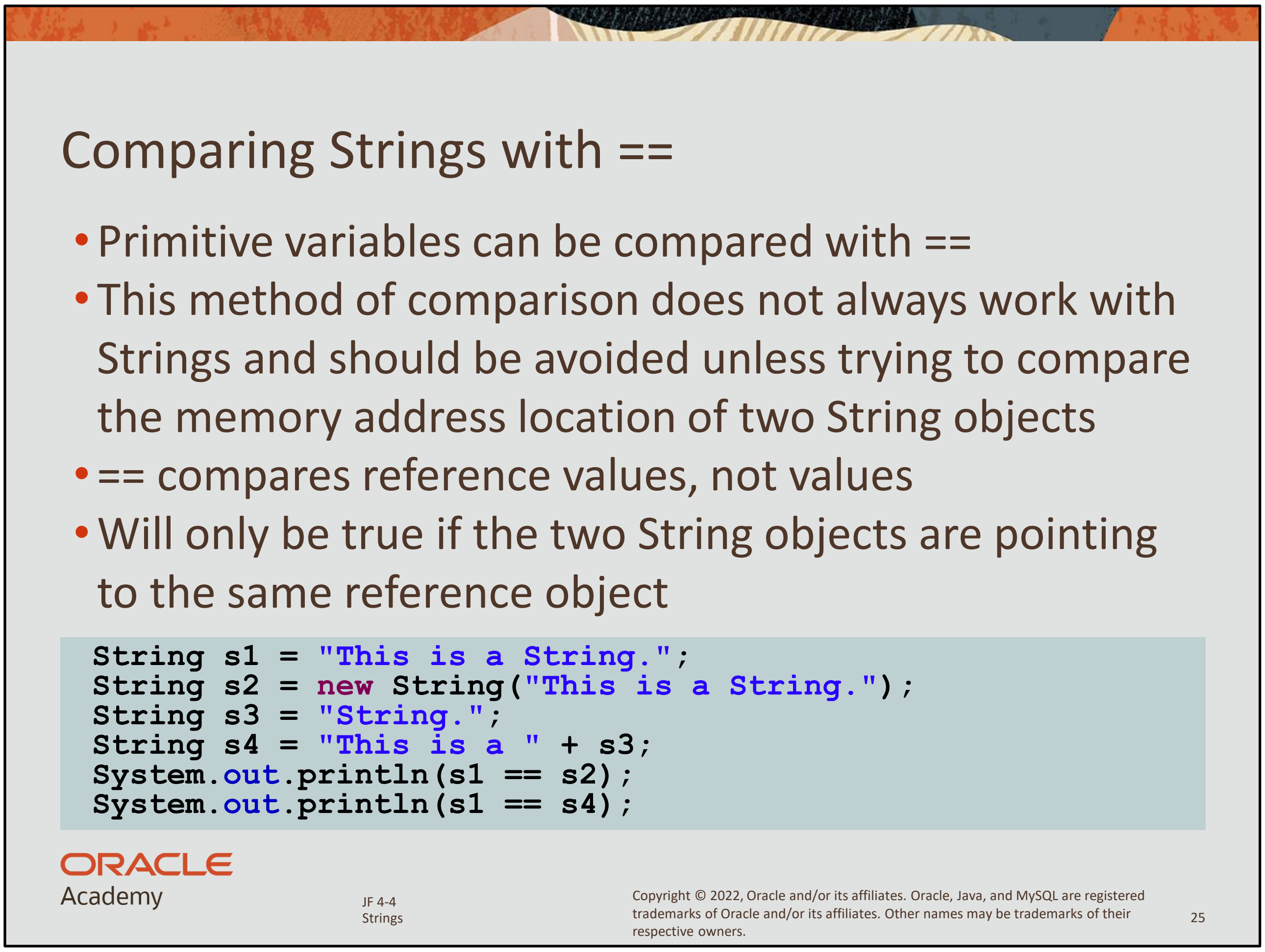
- This compareTo method prints an int less than 0

```
String s1 = "abc";
String s2 = "cde";
System.out.println(s1.compareTo(s2));
```

- This equals method prints false

```
String s1 = "abc";
String s2 = "ABC";
System.out.println(s1.equals(s2));
```

ORACLE
Academy



Comparing Strings with ==

- Primitive variables can be compared with ==
- This method of comparison does not always work with Strings and should be avoided unless trying to compare the memory address location of two String objects
- == compares reference values, not values
- Will only be true if the two String objects are pointing to the same reference object

```
String s1 = "This is a String.";
String s2 = new String("This is a String.");
String s3 = "String.";
String s4 = "This is a " + s3;
System.out.println(s1 == s2);
System.out.println(s1 == s4);
```

ORACLE
Academy

JF 4-4
Strings

25

Never use == to see if two String objects have the same value. This will produce logic errors, that is, code that compiles and runs but not give the correct answer all of the time.
== compares the addresses where the object are in memory, not the actual values at those location.
When using ==, you could have two String objects with the same value return a false.

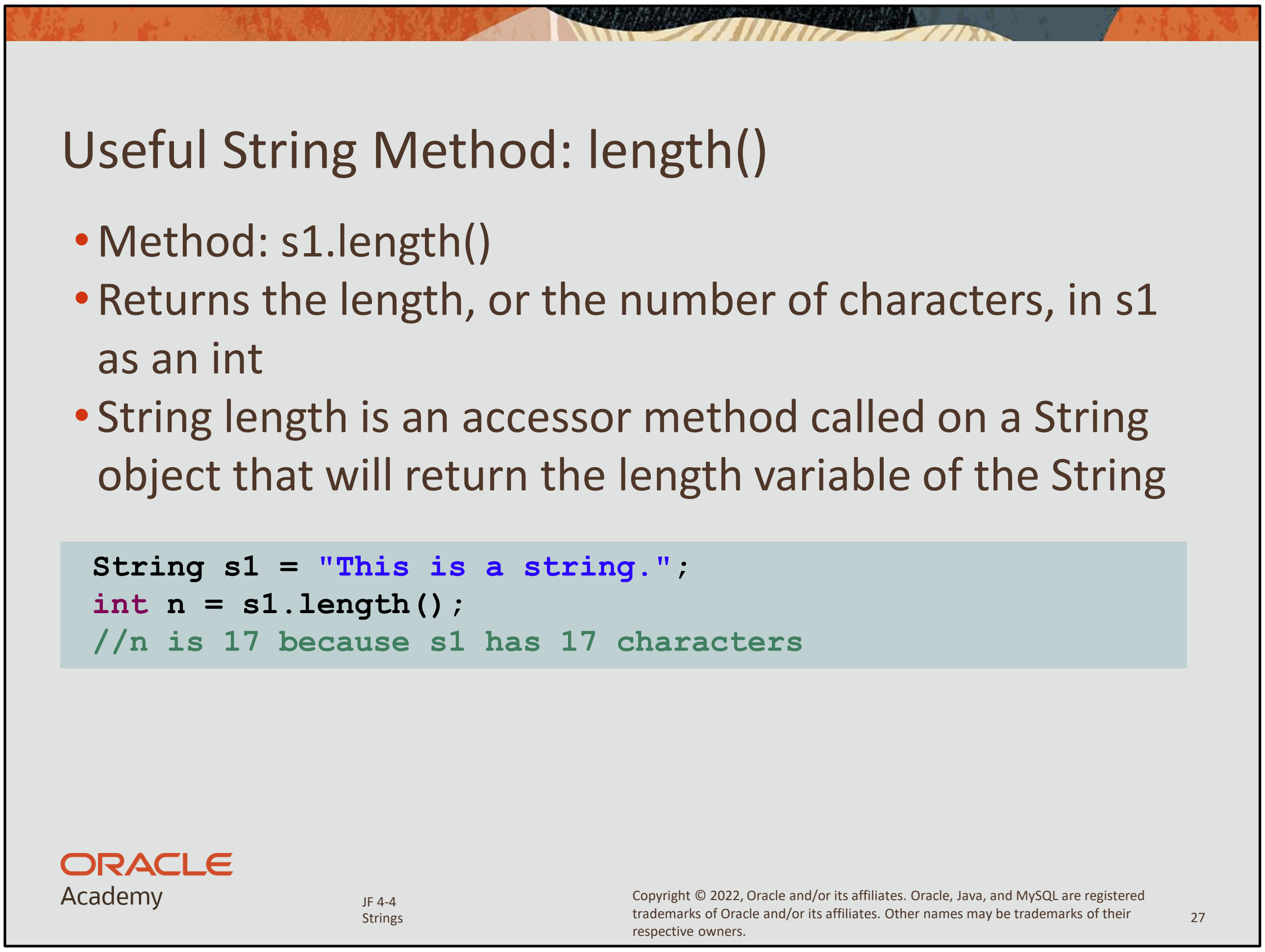
Comparing Strings with ==

- Which will print true in the example below?
  - s1 and s2 will point to the same object
  - Java notices that they are the same and makes the reference the same
  - s5 will not point to the same object as s1
  - It is "built" from a combination of two Strings, and therefore, a new object is created

```
String s1 = "This is a String.";
String s2 = "This is a String.";
String s3 = "This is ";
String s4 = "a String.";
String s5 = s3 + s4;

System.out.println(s1 == s2);
System.out.println(s1 == s5);
```

ORACLE Academy



Useful String Method: length()

- Method: s1.length()
- Returns the length, or the number of characters, in s1 as an int
- String length is an accessor method called on a String object that will return the length variable of the String

```
String s1 = "This is a string.";
int n = s1.length();
//n is 17 because s1 has 17 characters
```

ORACLE Academy

JF 4-4
Strings

27

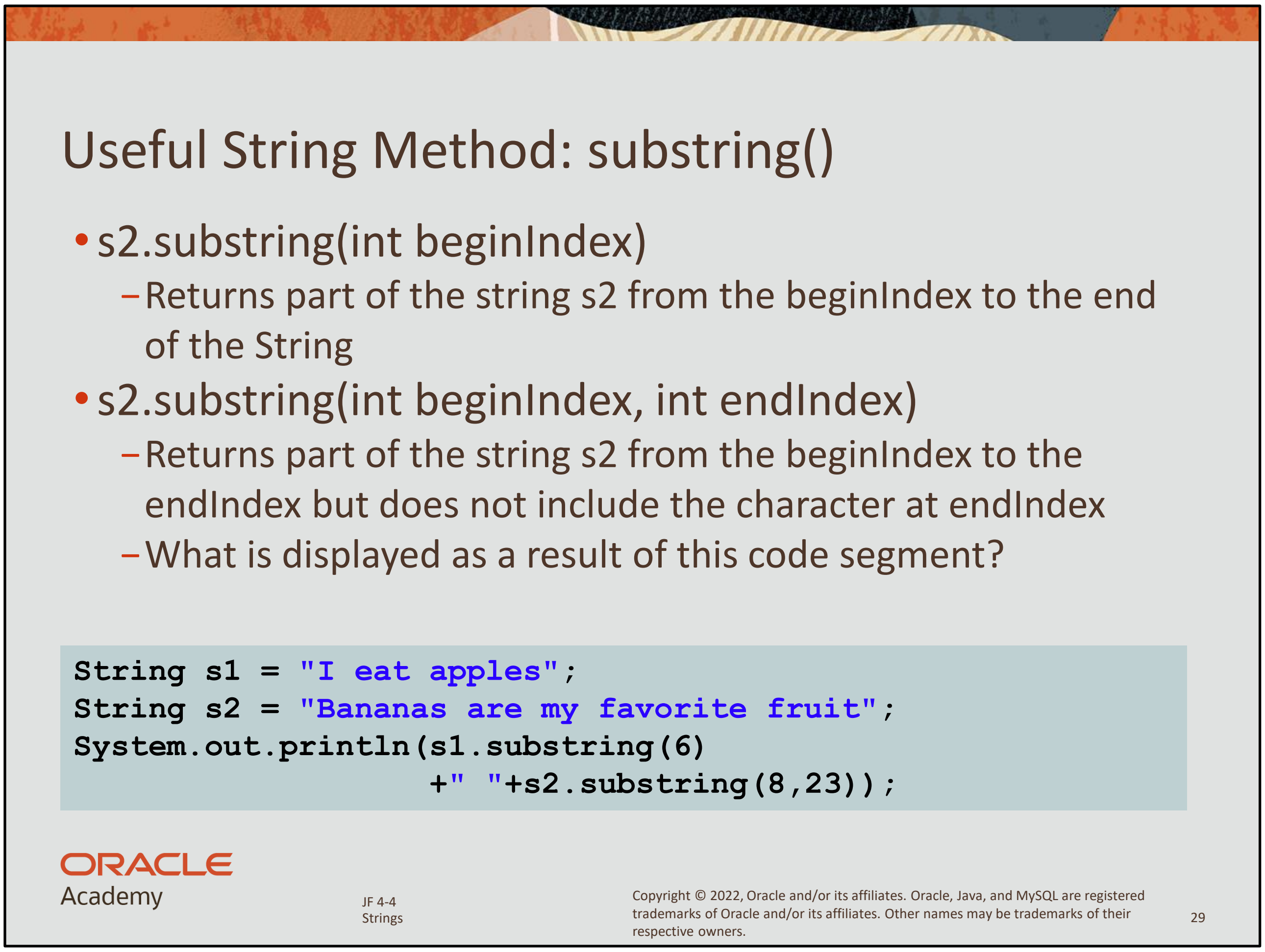
A string in Java can have a length of 0 - "". This is often called the empty string or the null string.

Each String Character Has an Index

- Since Strings are a representation of a sequence of characters, each character of a String has an index
- An index is a position reference
- The first character of a String has an index of 0, the second has an index of 1, and so on until the end of the String

ORACLE Academy

Conceptually, this is similar to Arrays, which will be covered later in this course.



Useful String Method: substring()

- s2.substring(int beginIndex)
  - Returns part of the string s2 from the beginIndex to the end of the String
- s2.substring(int beginIndex, int endIndex)
  - Returns part of the string s2 from the beginIndex to the endIndex but does not include the character at endIndex
  - What is displayed as a result of this code segment?

```
String s1 = "I eat apples";
String s2 = "Bananas are my favorite fruit";
System.out.println(s1.substring(6)
                   +" "+s2.substring(8,23));
```

ORACLE Academy

Useful String Methods: indexOf and charAt

- Method: **`s3.indexOf(char c);`**
  - Returns the index value of the first occurrence of c in String s3
- Parameter does not need to be a character, it can also be a String, for example: **`s3.indexOf("the");`**
- Method: **`s4.charAt(int index);`**
  - Returns the character of the String located at the index passed as the parameter
  - Index can be an integer from 0 to **`s4.length()-1`**

Using String Methods Example 1

- Write the Java code that will take an email address as input in the form of a String
  - This method will return the domain
  - For example, if the user enters john@oracle.com, the method returns oracle.com
  - To solve this problem, first search for the '@' character in the String
  - Then output the substring portion of the initial String that comes after this position

```
String domain="";          //no space between the " and "
int position=email.indexOf('@');
domain=email.substring(position+1);
System.out.println(domain);
```

ORACLE
Academy

Terminology

- Key terms used in this lesson included:
  - Concatenation
  - Escape sequences
  - Instantiate
  - Reference object
  - String object
  - String methods compareTo() and equals()

Summary

- In this lesson, you should have learned how to:
  - Instantiate (create) a String
  - Describe what happens when a String is modified
  - Use the + and += operators for concatenating Strings
  - Interpret escape sequences in String literals
  - Recognize the difference between a String and a primitive char data type
  - Test Strings with the compareTo() and equals() method
  - Describe why the == operator does not always work when testing String equality
  - Use String methods length(), substring(), indexOf(), and charAt()

ORACLE
Academy