More on Strings

String methods and equality

Produced by: Dr. Siobhán Drohan
Ms. Mairead Meagher

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

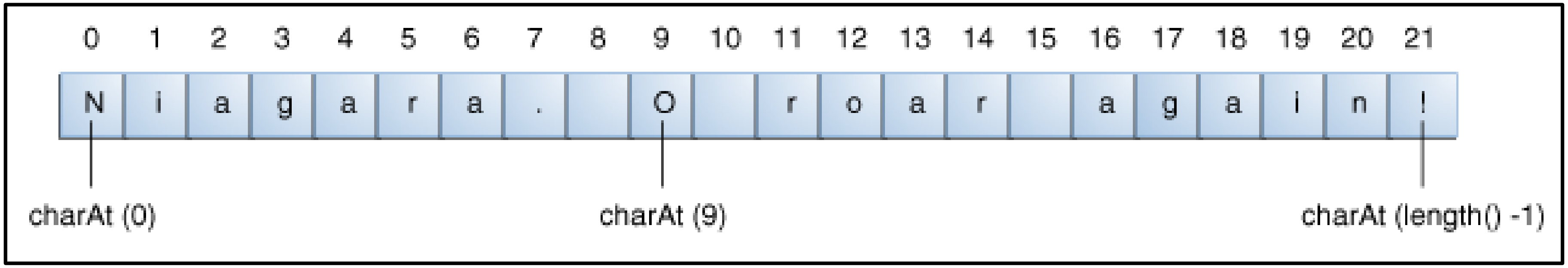
Department of Computing and Mathematics
http://www.wit.ie/

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

Strings: index of characters

- A String holds a sequence of characters

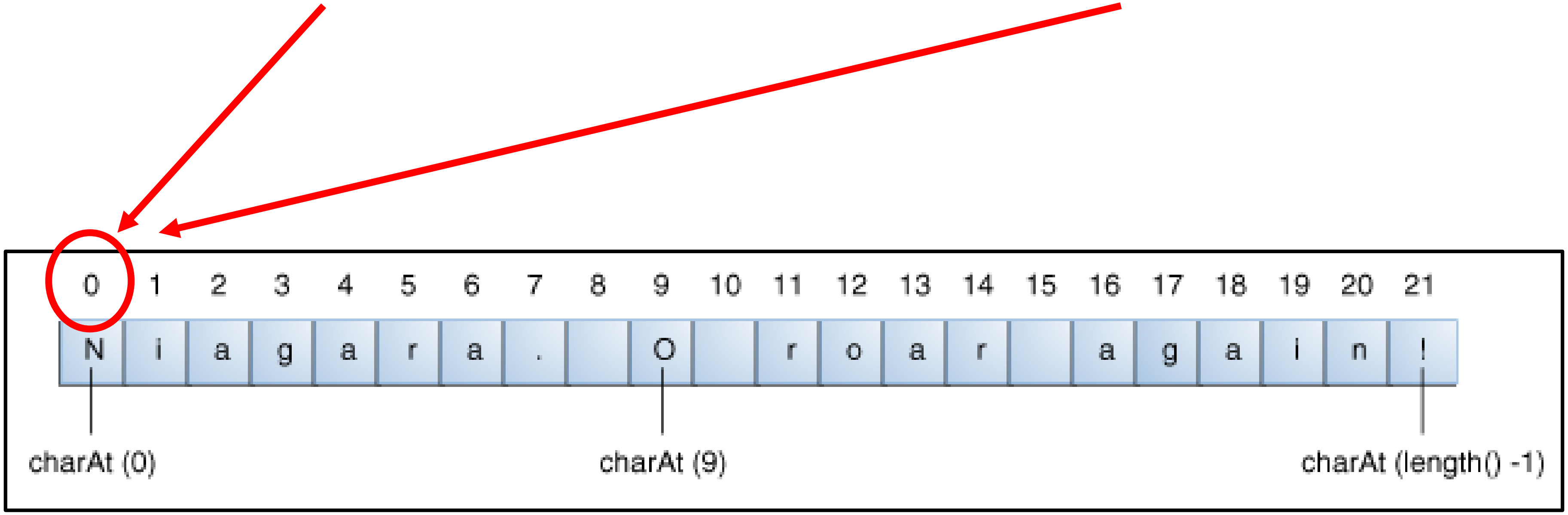


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

Strings: index of characters

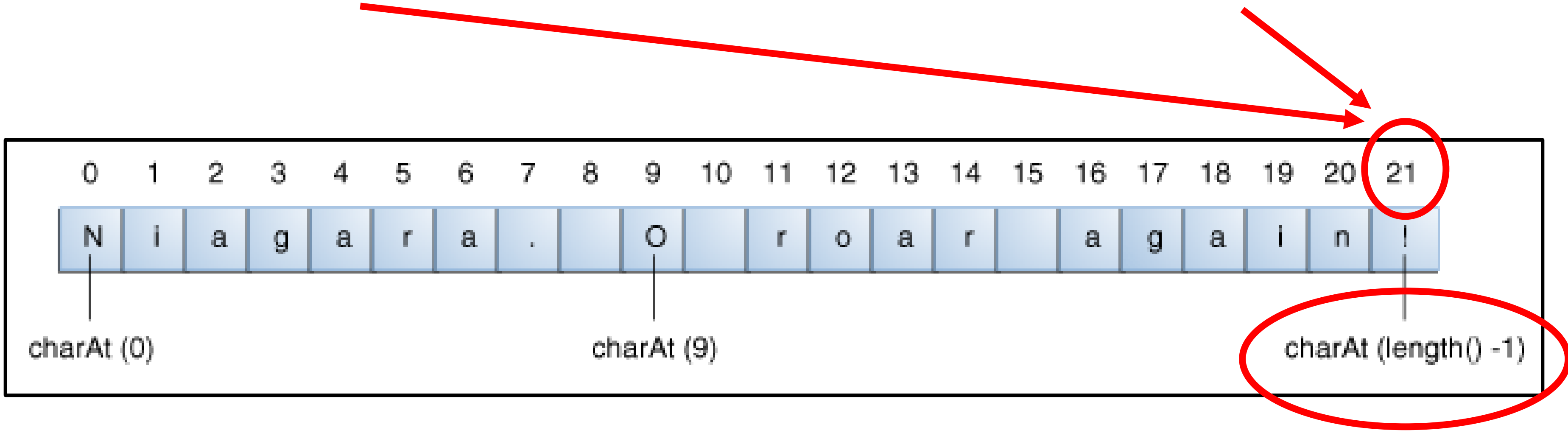
- A String holds a sequence of characters.
- **first character** in a String has an **index 0**



https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

Strings: index of characters

- A String holds a sequence of characters
- **first character** in a String has an **index 0**
- **last character** in a String has an **index length()-1**



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

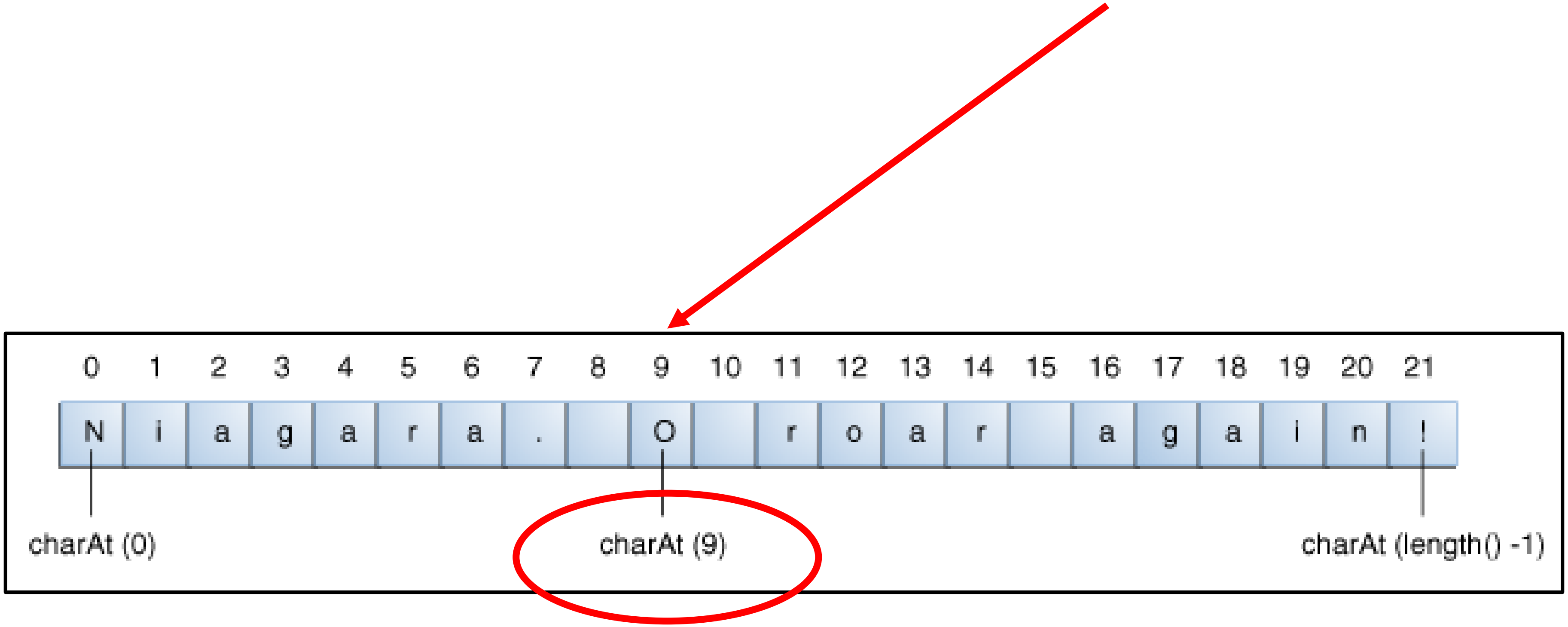
https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

Topics list

1. Strings: index of characters
2. **String methods**:
   - ➡ **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

String methods: **charAt** (int index)

- Say we want the character at index 9 in a String:



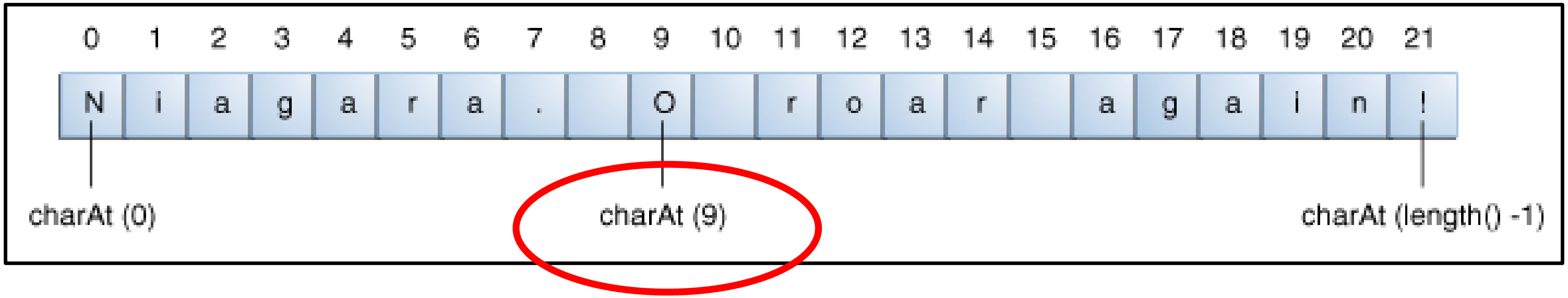
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods: **charAt** (int index)

- Say we want the character at index 9 in a String:

```
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```



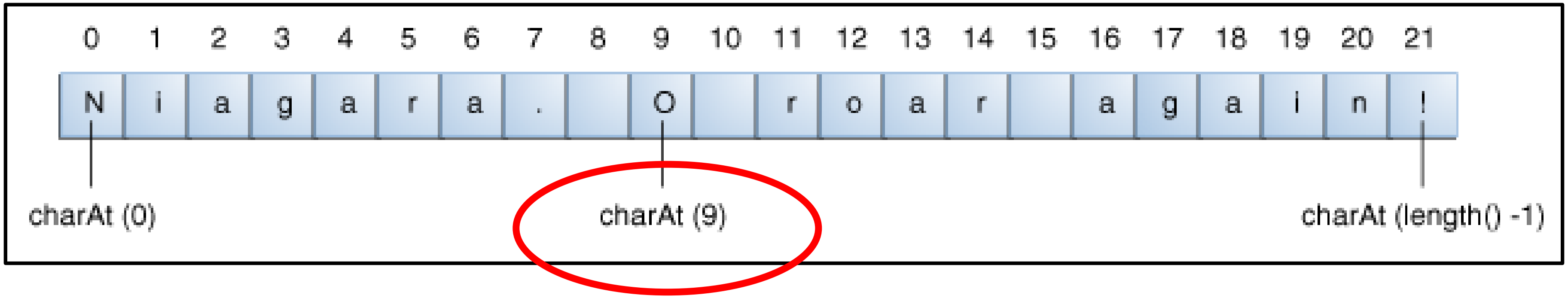
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods: **charAt** (int index)

- Say we want the character at index 9 in a String:

```
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

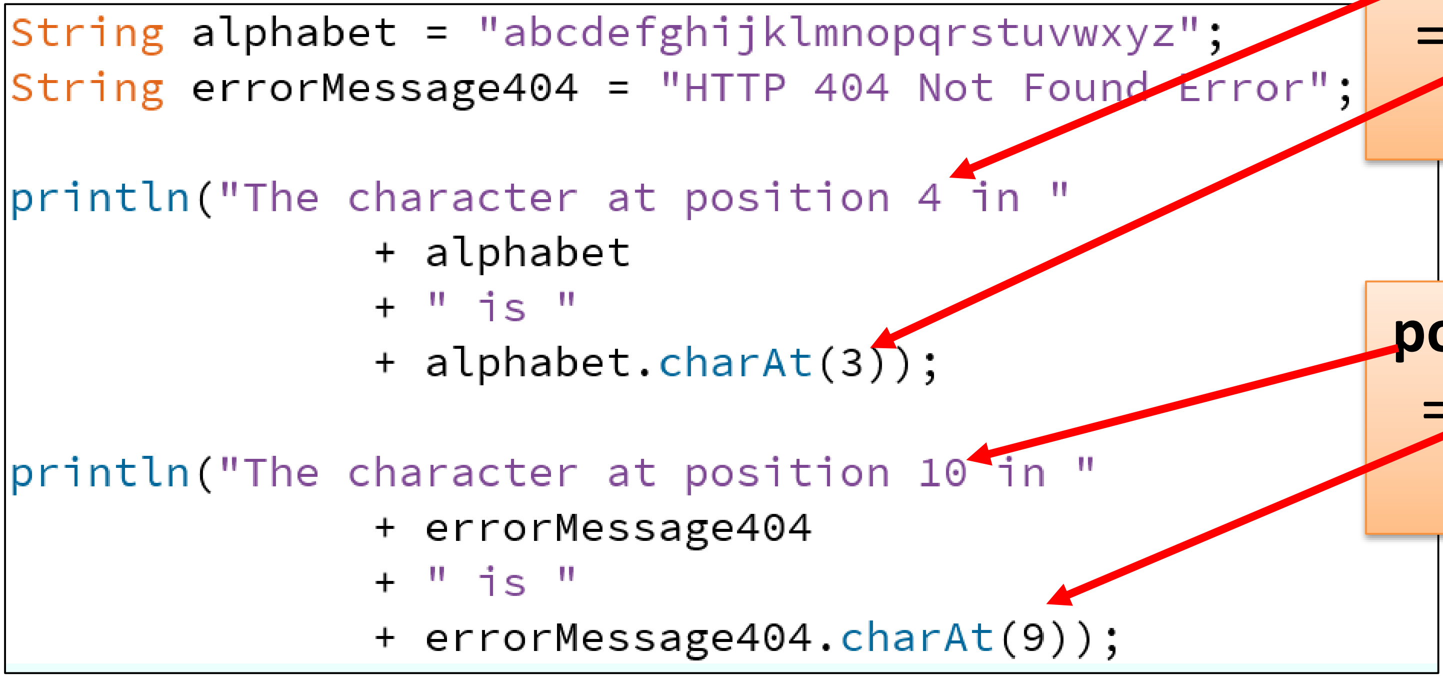


Indices begin at 0, so the character at index 9 is 'O' i.e. the 10th character

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

Example 7.1

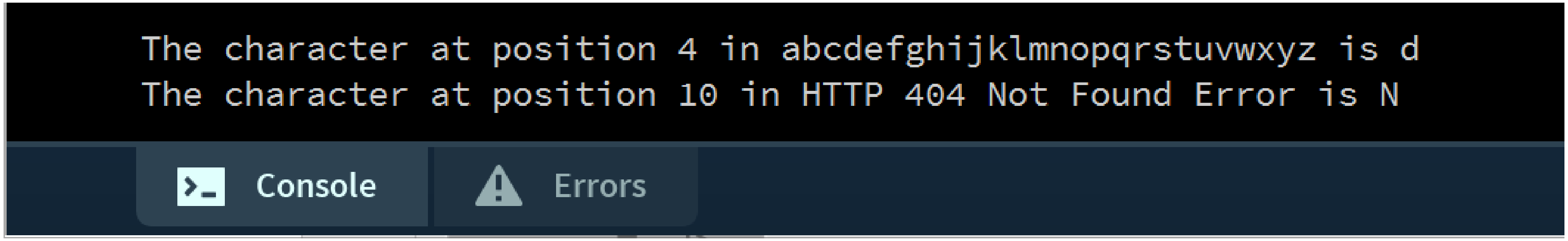
Finding the character located at specific **position** in a String.



```
String alphabet = "abcdefghijklmnopqrstuvwxyz";
String errorMessage404 = "HTTP 404 Not Found Error";

println("The character at position 4 in "
              + alphabet
              + " is "
              + alphabet.charAt(3));

println("The character at position 10 in "
              + errorMessage404
              + " is "
              + errorMessage404.charAt(9));
```

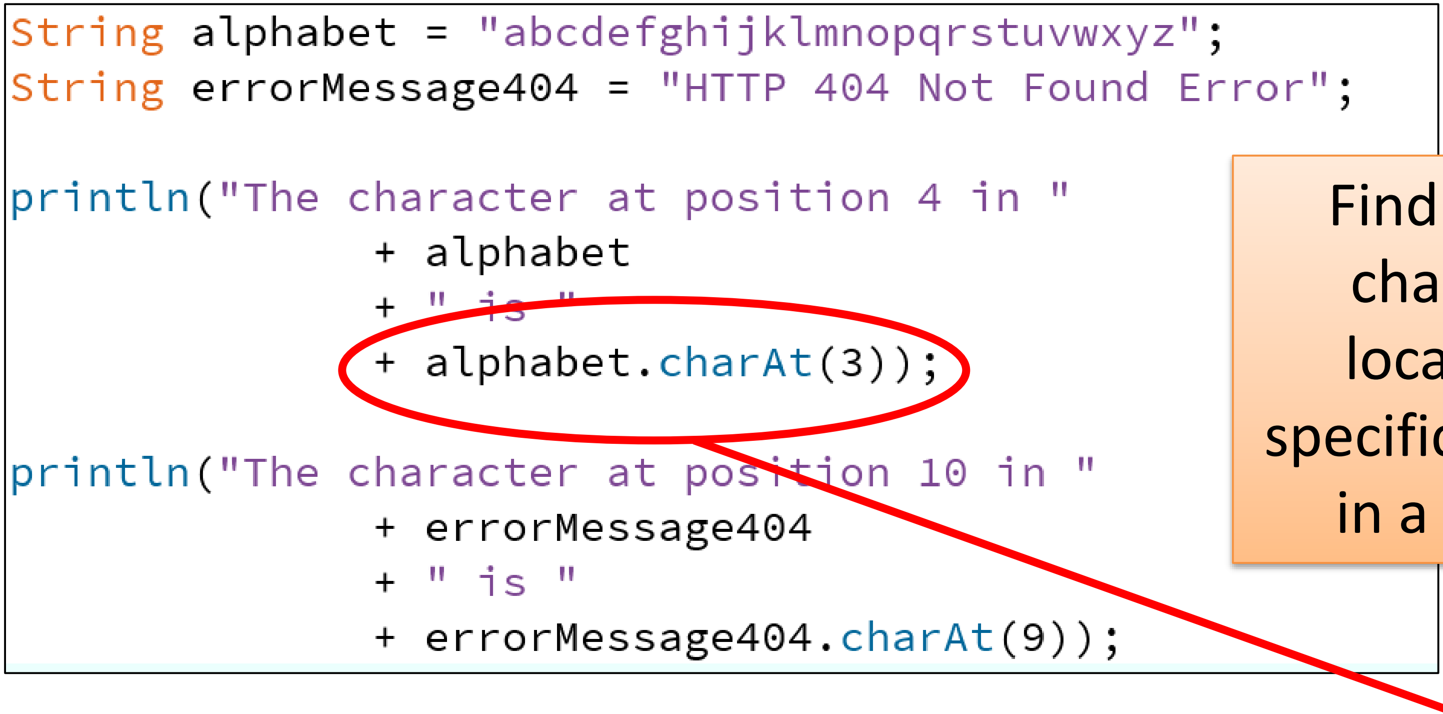
position 4
= **index** 3
= **d**

position 10
= **index** 9
= **N**



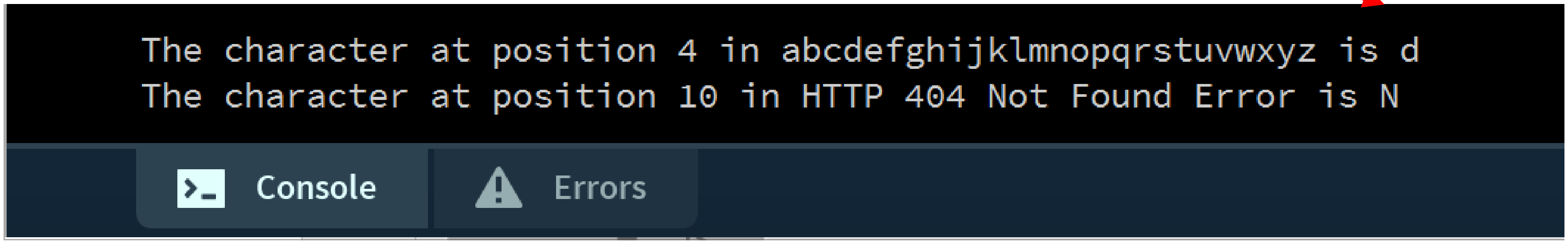
```
The character at position 4 in abcdefghijklmnopqrstuvwxyz is d
The character at position 10 in HTTP 404 Not Found Error is N
```

Example 7.1



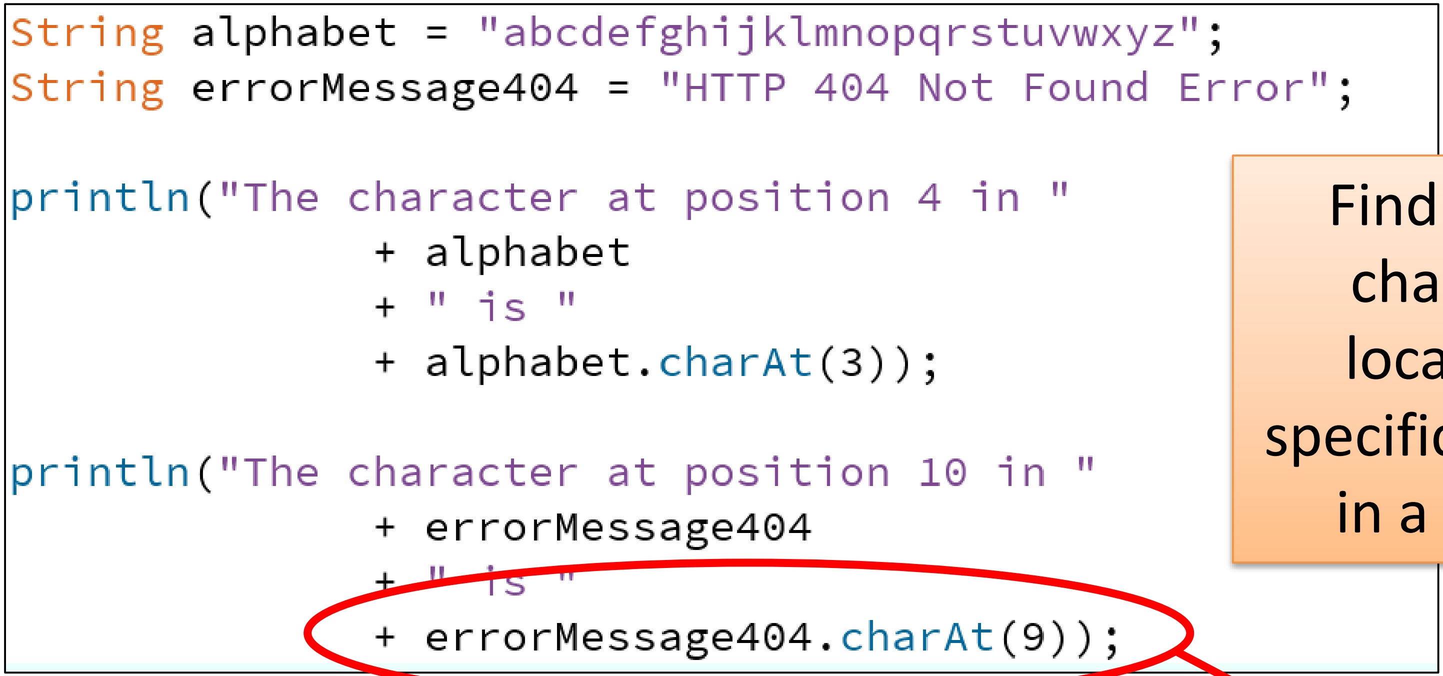
```
String alphabet = "abcdefghijklmnopqrstuvwxyz";
String errorMessage404 = "HTTP 404 Not Found Error";

println("The character at position 4 in "
              + alphabet
              + " is "
              + alphabet.charAt(3));

println("The character at position 10 in "
              + errorMessage404
              + " is "
              + errorMessage404.charAt(9));
```

Finding the character located at specific position in a String.



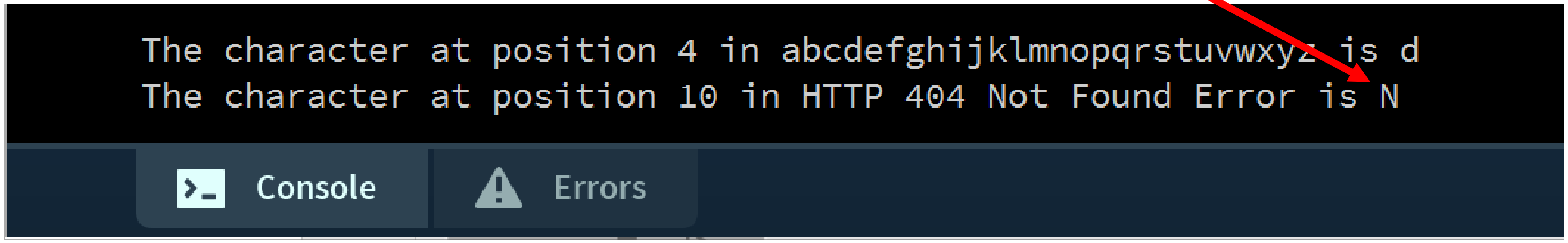
```
The character at position 4 in abcdefghijklmnopqrstuvwxyz is d
The character at position 10 in HTTP 404 Not Found Error is N
```

Example 7.1



```
String alphabet = "abcdefghijklmnopqrstuvwxyz";
String errorMessage404 = "HTTP 404 Not Found Error";

println("The character at position 4 in "
              + alphabet
              + " is "
              + alphabet.charAt(3));

println("The character at position 10 in "
              + errorMessage404
              + " is "
              + errorMessage404.charAt(9));
```

Finding the character located at specific position in a String.



```
The character at position 4 in abcdefghijklmnopqrstuvwxyz is d
The character at position 10 in HTTP 404 Not Found Error is N
```

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - ➡ **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

String methods:
substring (int beginIndex, int endIndex)

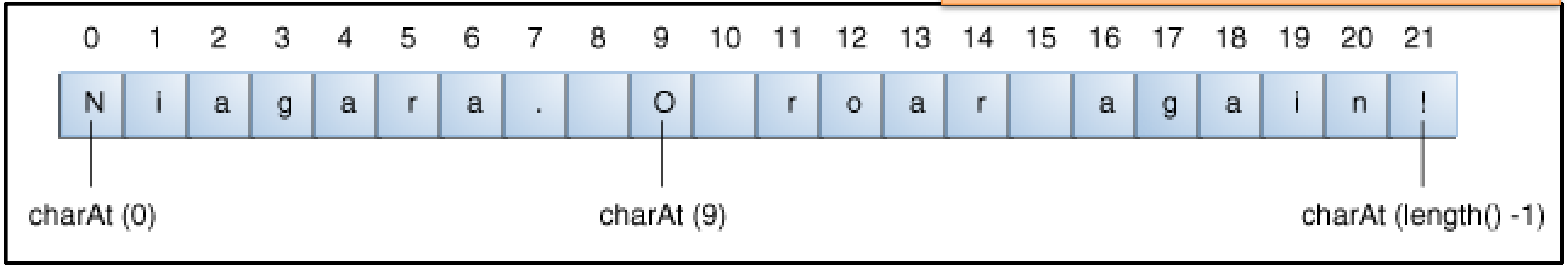
- This method returns a new String that is a substring of this String.

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods:
substring (int beginIndex, int endIndex)

- This method returns a new String that is a substring of this String.

Given this String…

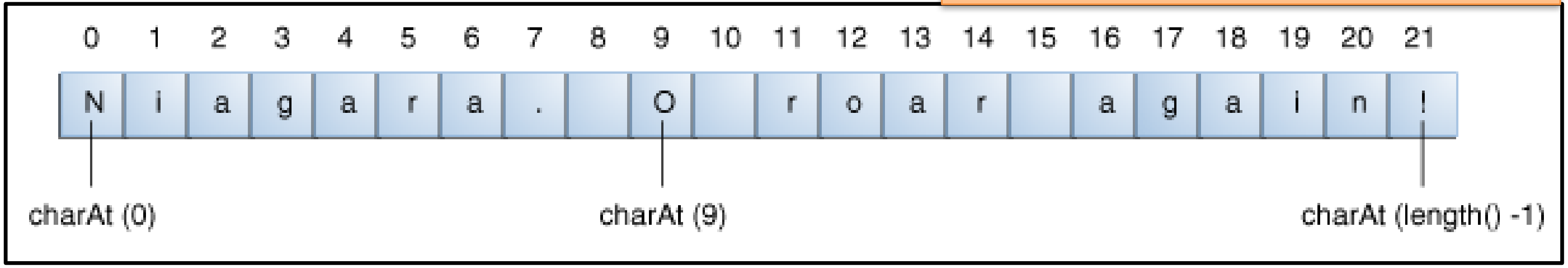


https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods:
substring (int beginIndex, int endIndex)

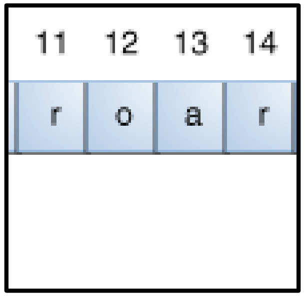
- This method returns a new String that is a substring of this String.

Given this String…



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

…this is a substring →



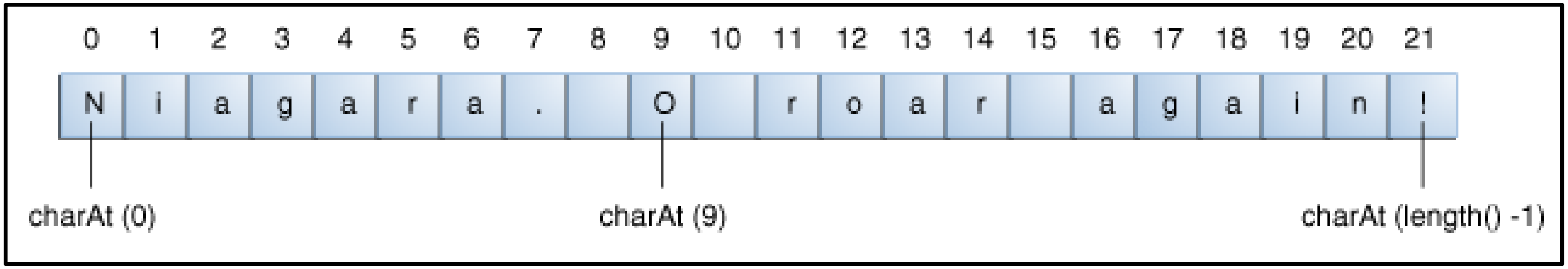
11	12	13	14
r	o	a	r

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

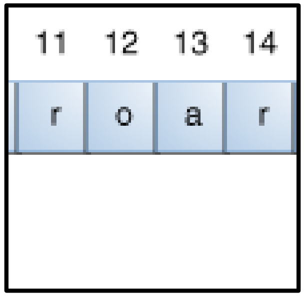
String methods:
substring (int beginIndex, int endIndex)

The substring begins at the specified **beginIndex…**

…and extends to the character at index **endIndex-<u>1</u>**

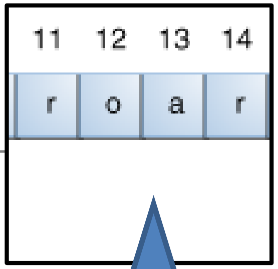


…this is a substring →



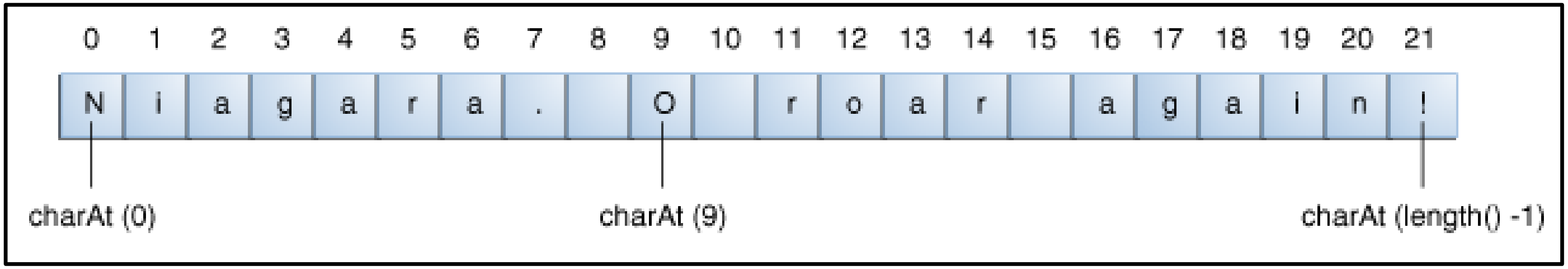
https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods:
substring (int beginIndex, int endIndex)



The substring begins at the specified **beginIndex…**

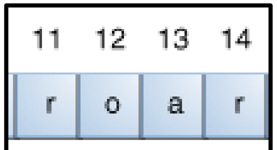
…and extends to the character at index **endIndex-1**



String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.**substring**(11, 15);

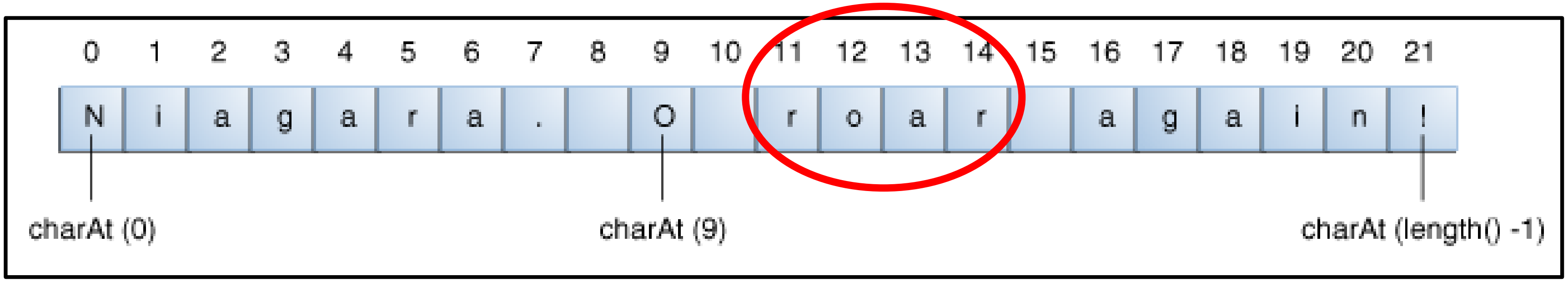
https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

String methods:
substring (int beginIndex, int endIndex)



This code returns a substring (“roar”) from anotherPalindrome.

It extends from index **11** up to **15 -1**, i.e. 11,12,13,14



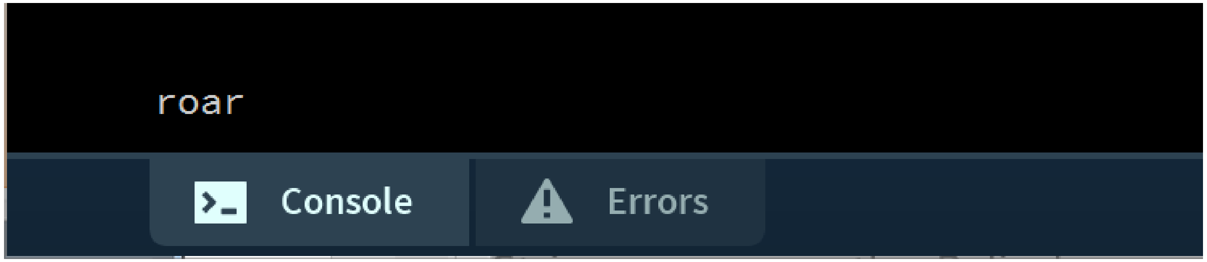
```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```

https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html

Example 7.2, **version 1**

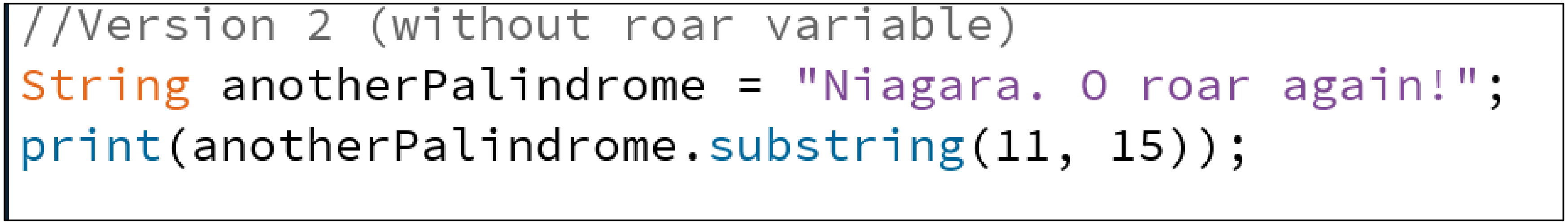
```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
print(roar);
```

Printing out a substring of a String to the console.



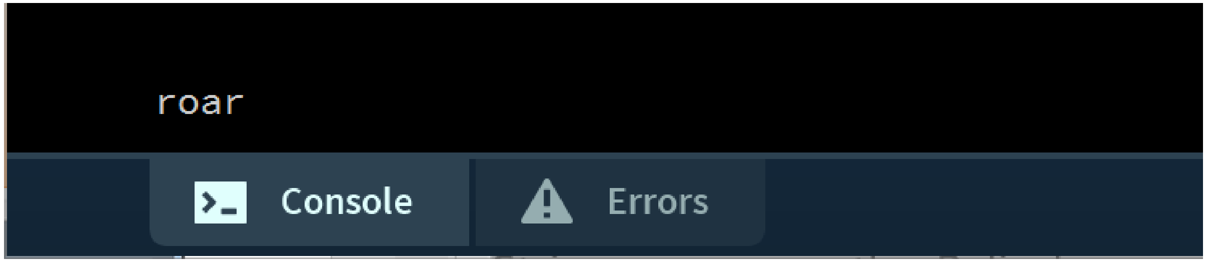
```
roar
```

Example 7.2, **version 2**



```
//Version 2 (without roar variable)
String anotherPalindrome = "Niagara. O roar again!";
print(anotherPalindrome.substring(11, 15));
```

Printing out a substring of a String to the console.



```
roar
```

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - ➡ **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

String methods: **compareTo**

int **compareTo** (String anotherString)

- This method compares two strings **lexicographically**
  - i.e.
    based on the Unicode value of the characters in the String.

- It returns an integer indicating whether this string is:
  - greater than (result is **> 0**)
  - equal to (result is **= 0**) or
  - less than (result is **< 0**)

  the argument, anotherString.

https://docs.oracle.com/javase/tutorial/java/data/comparestrings.html

Examples 7.3 - 7.6

- In the next 4 examples we compare 2 strings
  str1**.compareTo**(str2)
- where str2 = "Cat"
- And str1 =
  - "Dog"
  - then "cat"
  - then "Animal"
  - then "Cat"

Example **7.3 – Dog**

Q: What will be printed to the console?

Q: Which boolean expression evaluates to true?

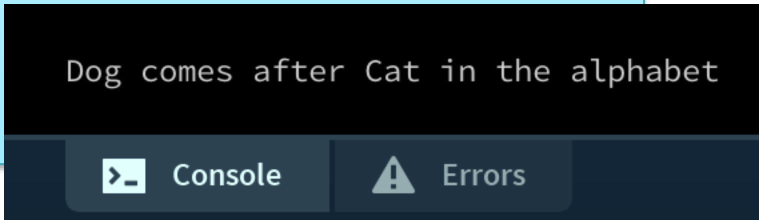
```
String str1 = "Dog";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)            { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)    { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

Example **7.3**

A: str1.compareTo (str2)

returns a positive integer
as “Dog” (str1) comes **after** “Cat” (str2).

```
String str1 = "Dog";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)          { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)     { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```



```
Dog comes after Cat in the alphabet
```

Example **7.4 - cat**

Q: What will be printed to the console?

Q: Which boolean expression evaluates to true?

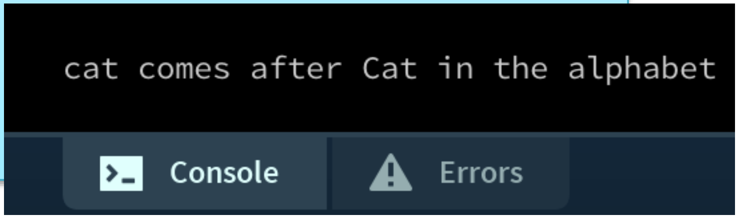
```
String str1 = "cat";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)          { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)    { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

Example 7.4

A: str1.compareTo(str2)

returns a positive integer
as “cat” (str1) comes after “Cat” (str2)
in the Unicode character map.

```
String str1 = "cat";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)           { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)     { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
    println ("The strings are identical");
}
```



```
cat comes after Cat in the alphabet
```

Example **7.5 - Animal**

Q: What will be printed to the console?

Q: Which boolean expression evaluates to true?

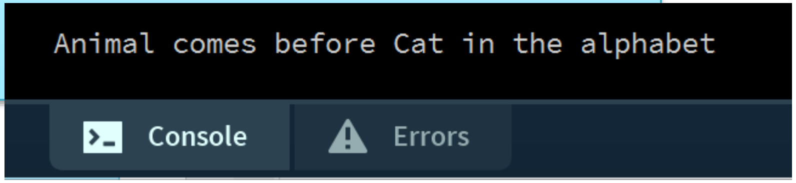
```
String str1 = "Animal";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)          { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)    { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

Example **7.5**

```
String str1 = "Animal";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)           { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)     { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

A: str1.compareTo(str2)

returns a negative integer
as Animal(str1) comes before Cat (str2)
in the Unicode character map.



```
Animal comes before Cat in the alphabet
```

Example **7.6 - Cat**

Q: What will be printed to the console?

Q: Which boolean expression evaluates to true?

```
String str1 = "Cat";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)          { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)     { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

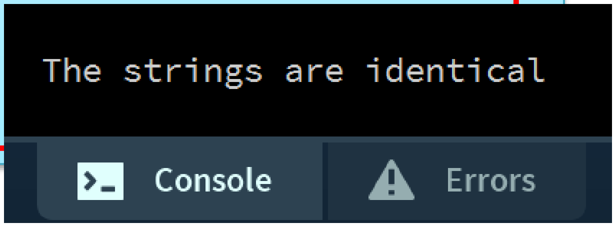
Example **7.6**

```
String str1 = "Cat";
String str2 = "Cat";

if (str1.compareTo(str2) < 0)          { // before
    println(str1+" comes before "+ str2 +" in the alphabet");
}
else if (str1.compareTo(str2) > 0)    { // after
    println (str1 +" comes after "+ str2  +" in the alphabet");
}
else{
   println ("The strings are identical");
}
```

A: str1.compareTo(str2)

returns 0

as Cat (str1) is identical to Cat (str2).



```
The strings are identical
```

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

Primitive types vs. Object types

Primitive type

```
int i = 17;
```

Primitive types vs. Object types

Primitive type

```
int i = 17;
```

Directly stored in memory…

17

Primitive types vs. Object types

Primitive type

```
int i = 17;
```

Directly stored in memory…

17

Object type

```
String hi = “Hello”;
```

Primitive types vs. Object types

Primitive type

```
int i = 17;
```

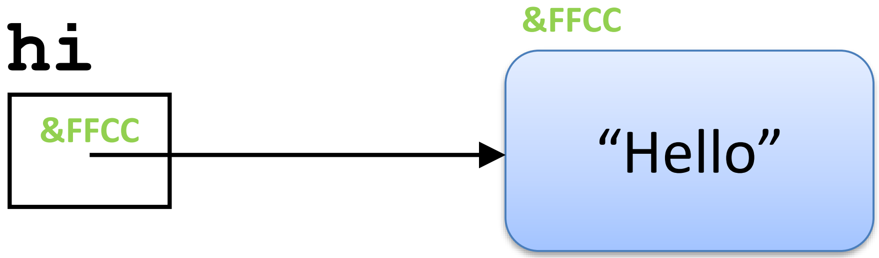
Directly stored in memory…

17

Object type

```
String hi = “Hello”;
```

hi variable contains a **reference** (*address*) to where the String is stored in memory



Primitive types vs. Object types

Primitive type

```
int i = 17;
```

Directly stored in memory…

17

With **primitive** type variables (e.g. int, float, char, etc)

the **value** of the variable is stored in the memory location assigned to the variable.

Primitive types vs. Object types

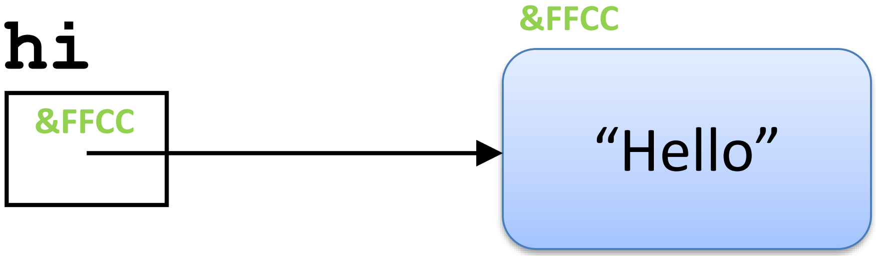
With **object** types, the variable holds the **memory address** of where the object is located – **not the values** inside the object.

This memory address is called a **reference** to the object.

Object type

```
String hi = “Hello”;
```

hi variable contains a reference (*address*) to where the String is stored in memory



Primitive types vs. Object types

Now that we know how primitive types and object types store data,

we will look at this statement (b=a) in the context of primitive and object types.

```
b = a;
```

Primitive types vs. Object types

Primitive types

```
b = a;
```

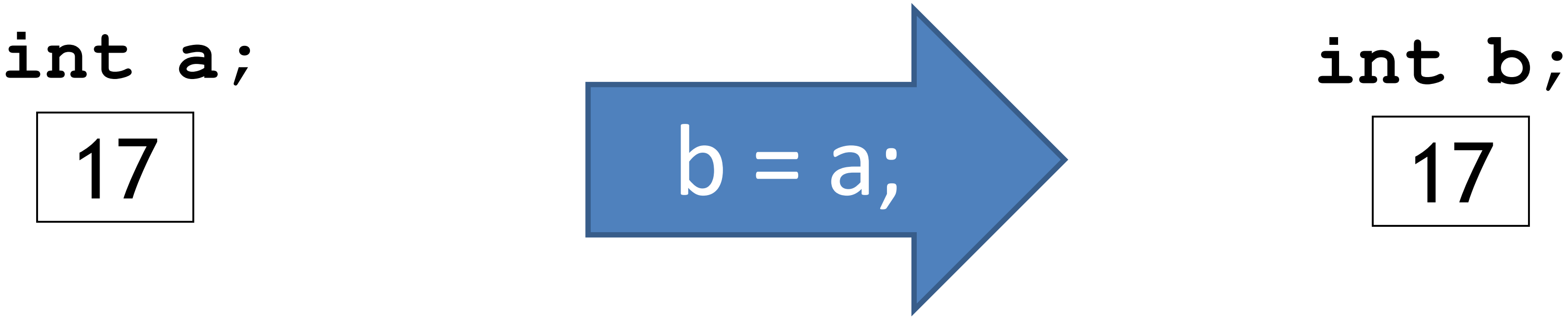
```
int a;
```

17

Primitive types vs. Object types

Primitive types

`b = a;`



```
int a;
```

17

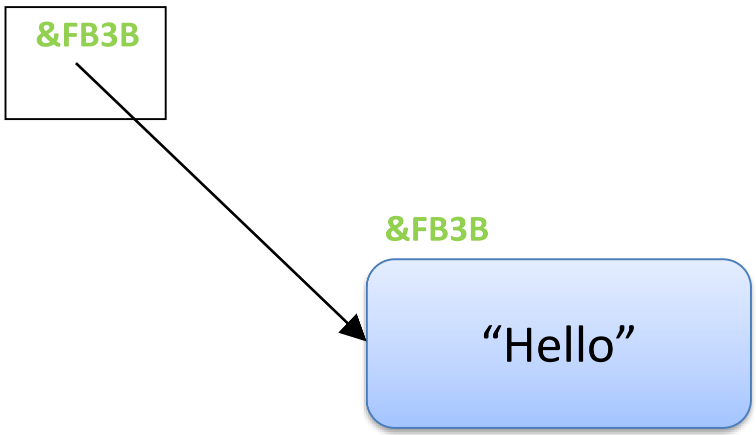
b = a;

```
int b;
```

17

Primitive types vs. Object types

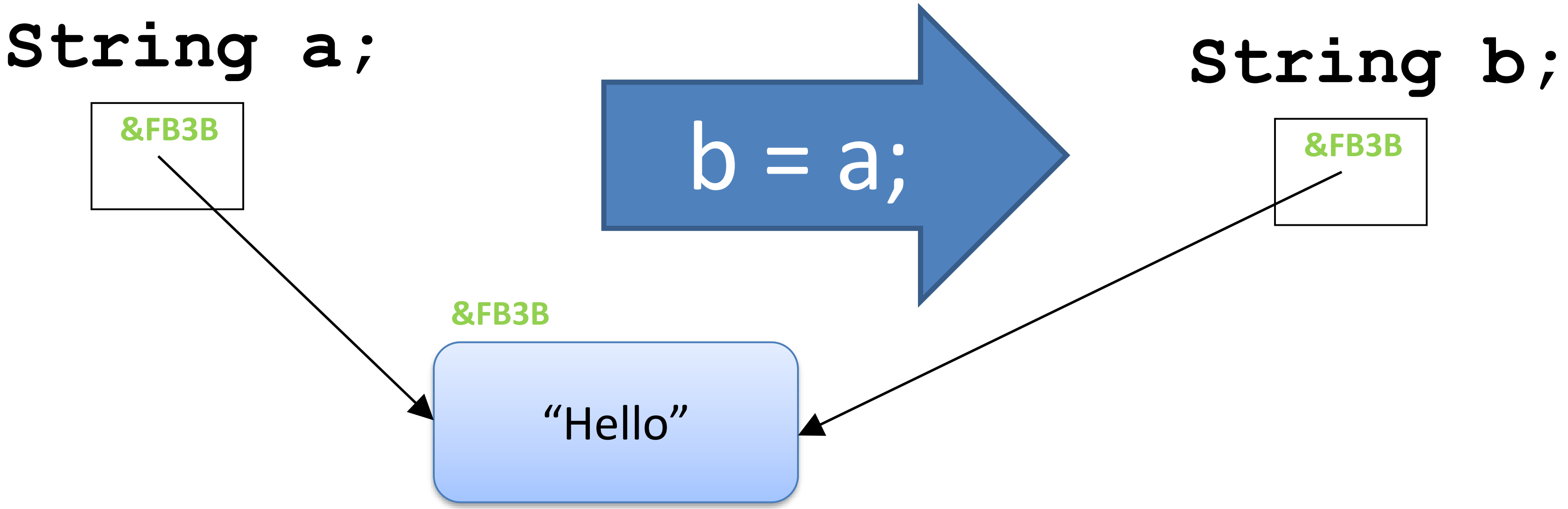
```
String a;
```



```
b = a;
```

Object types

Primitive types vs. Object types



```
String a;
```

&FB3B

b = a;

```
String b;
```

&FB3B

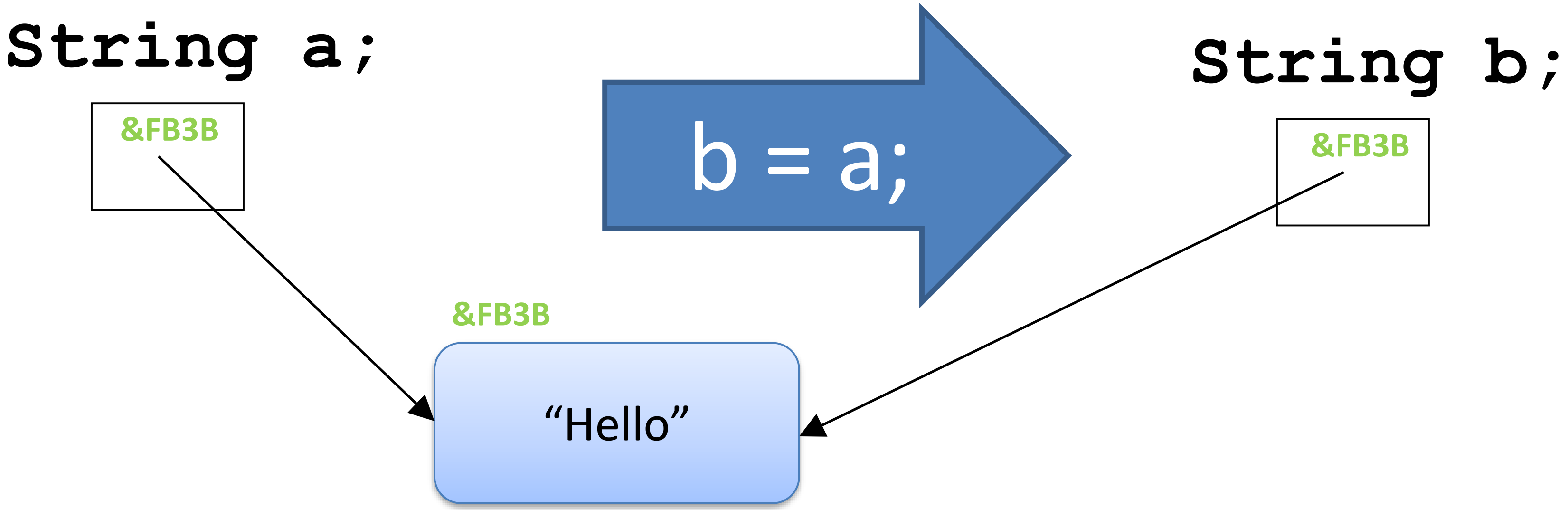
&FB3B

“Hello”

```
b = a;
```

Object types

Primitive types vs. Object types



String a;

&FB3B

b = a;

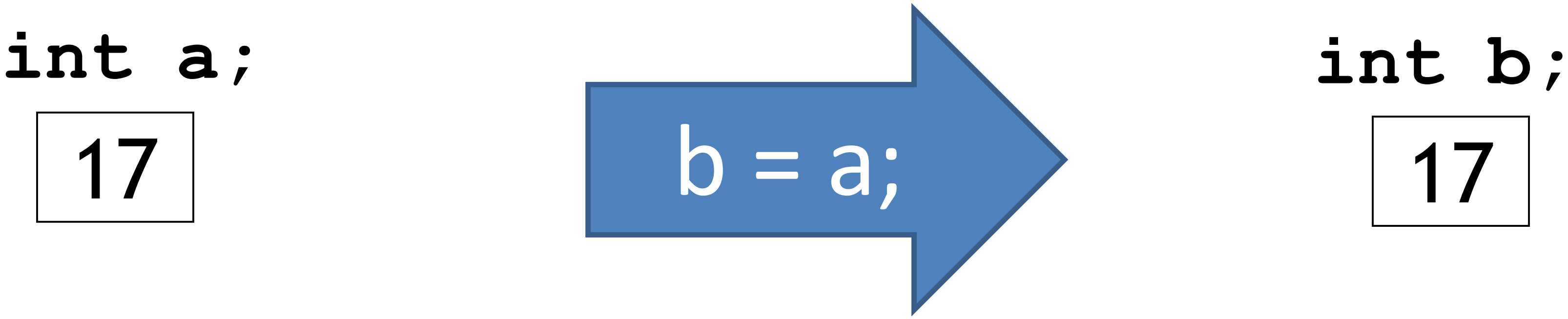
String b;

&FB3B

&FB3B

“Hello”

b = a;



int a;

17

b = a;

int b;

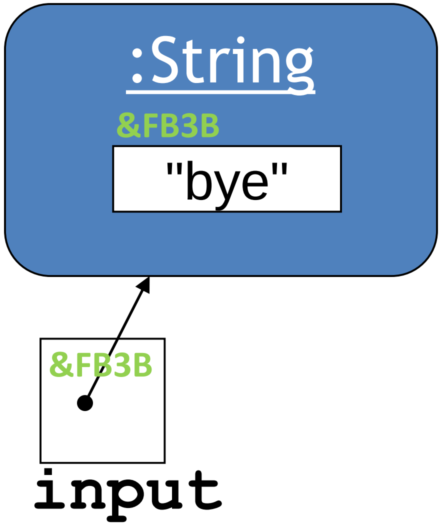
17

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

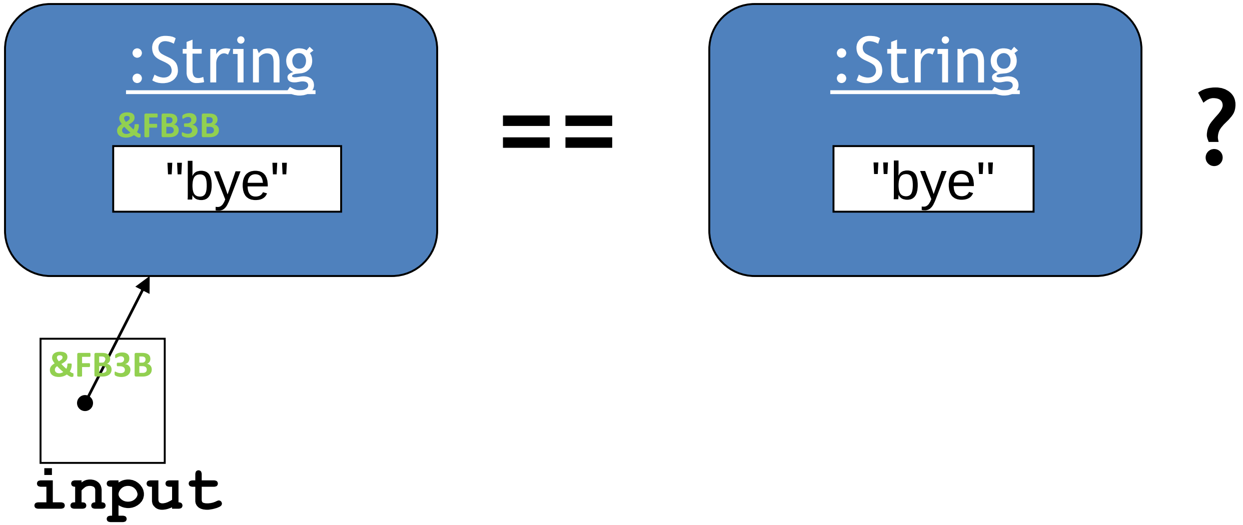
String: Identity vs Equality

```
String input = "bye";
```



String: Identity vs Equality

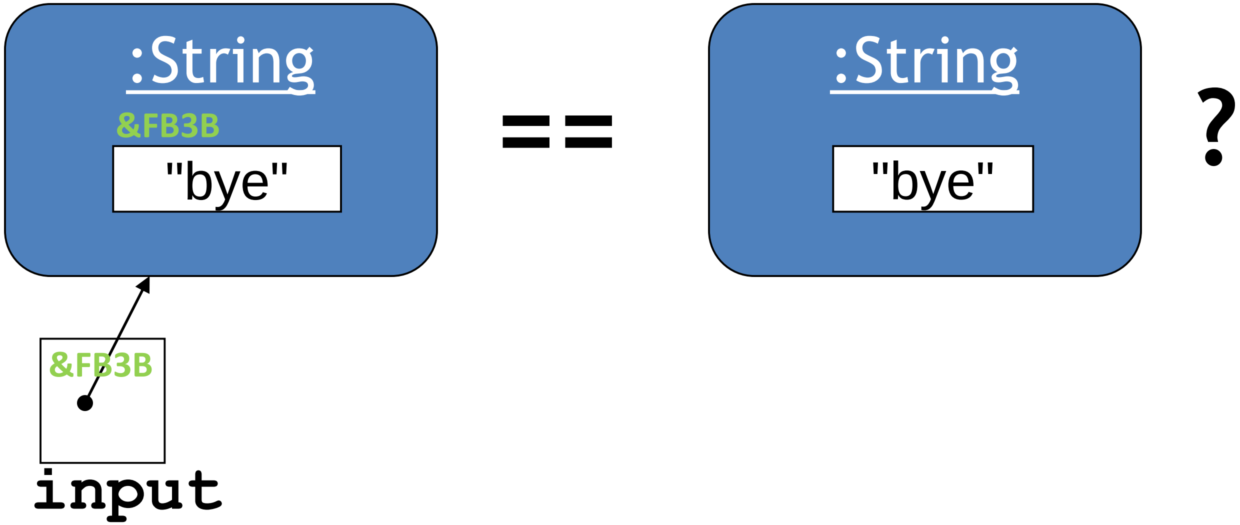
```
String input = "bye";
if(input == "bye") {
   //...
}
```



String: Identity vs Equality

```
String input = "bye";
if(input == "bye") {
   //...
}
```

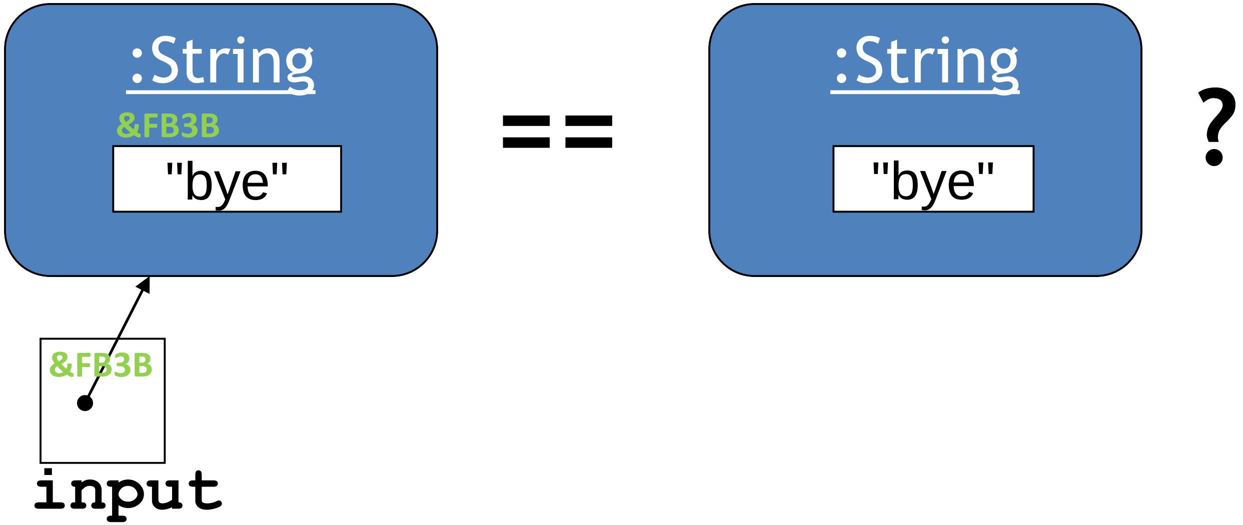
== tests identity



String: **Identity** vs Equality

```
String input = "bye";
if(input == "bye") {
    //...
}
```

== tests identity

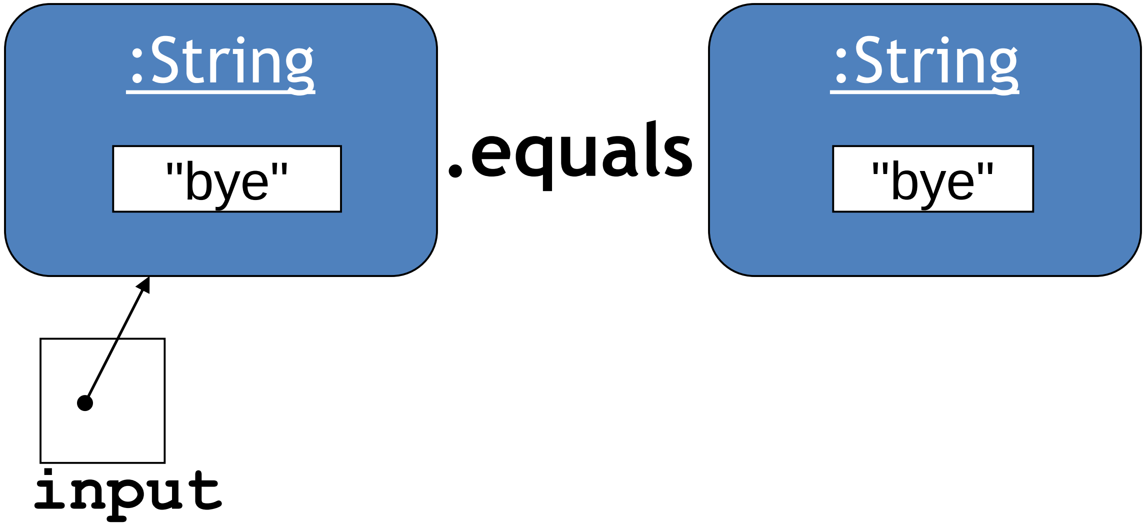


Answer: (maybe) false!

String: Identity vs **Equality**

```
String input = "bye";
if(input.equals("bye")) {
    ...
}
```

.equals tests equality



:String

"bye"

.equals

:String

"bye"

input

Answer: true

“bye” equals “bye”

String: **Identity vs Equality**

```
if(input == "bye") {
     ...
}


if(input.equals("bye")) {
     ...
}
```

tests **identity**
i.e. the reference

tests **equality**
i.e. string value

Strings should always be compared
using the **`.equals`** method

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

Q1: What’s wrong here?

```
void anyMethod()
{
    String str1 = "a";
    String str2 = "b";

    if(str1 == str2)
    {
        println(str1+" is the same as "+ str2);
    }
    else
    {
        println(str1+" is NOT same as "+ str2);
    }
}
```

A1: Strings need to use the .equals method

```
void anyMethod()
{
    String str1 = "a";
    String str2 = "b";

    if(str1 == str2)
    {
        println(str1+" is the same as "+ str2);
    }
    else
    {
        println(str1+" is NOT same as "+ str2);
    }
}
```

Q2: What’s wrong here?

```
public void anyMethod()
{
   int num1 = 1;
   int num2 = 2;

   if(num1 = num2)
      println(num1+" is the same as "+ num2);
   else
      println(num1+" is NOT same as "+ num2);
}
```

A: You need two equals for equality

```
public void anyMethod()
{
   int num1 = 1;
   int num2 = 2;

   if(num1 = num2)
      println(num1+" is the same as "+ num2);
   else
      println(num1+" is NOT same as "+ num2);
}
```

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

null

- `null` is a special value in Java.
- All object variables are initialised to `null`.

null

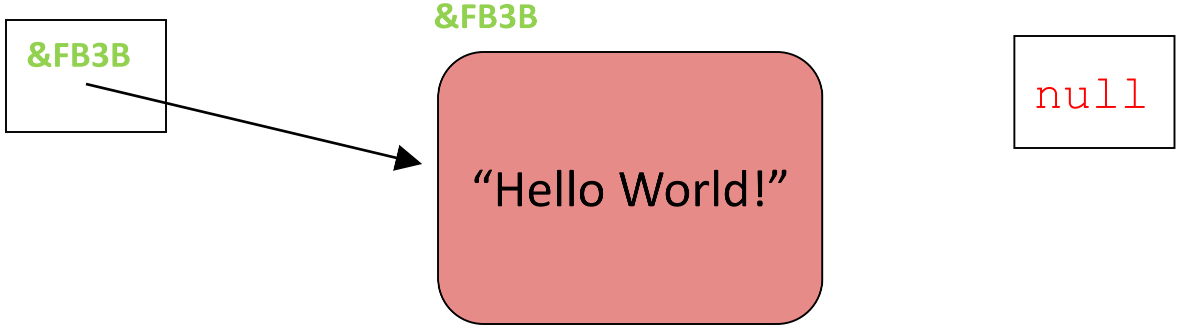
- null means that the object variable does not have a reference

  e.g.

  - str1 below has a reference to the string “hello World!”
  - str2 below does not have a reference. It is null.

```
String str1;
```

```
String str2;
```



null

You can assign and test for null:

```
String hours;

if(hours == null)
{
    //...
}
hours = null;
```

Topics list

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

Escape sequences

When a String is printed,
certain **single characters that follow a backslash (\)**
have special meaning…

…and the compiler interprets them accordingly.

http://docs.oracle.com/javase/tutorial/java/data/characters.html

Java escape sequences

Escape Sequence	**Description**
\t	Insert a **tab** in the text at this point.
\b	Insert a **backspace** in the text at this point.
\n	Insert a **newline** in the text at this point.
\r	Insert a **carriage return** in the text at this point.
\f	Insert a **formfeed** in the text at this point.
\'	Insert a **single quote** character in the text at this point.
\"	Insert a **double quote** character in the text at this point.
\	Insert a **backslash** character in the text at this point.

http://docs.oracle.com/javase/tutorial/java/data/characters.html

Examples of escape sequences

```
print(“Java\n”);
```

is the exact same as:

```
println(“Java”);
```

```
println(“	Java”);
```

is similar to:

```
println(“\tJava”);
```

http://docs.oracle.com/javase/tutorial/java/data/characters.html

Summary

1. Strings: index of characters
2. **String methods**:
   - **charAt**(int index)
   - **substring** (int beginIndex, int endIndex)
   - **compareTo** (String anotherString)
3. Recap: Primitive vs Object
4. **String identity** vs **equality**
5. Common **Errors** with Strings
6. **null**
7. **Escape Sequences**

Questions?