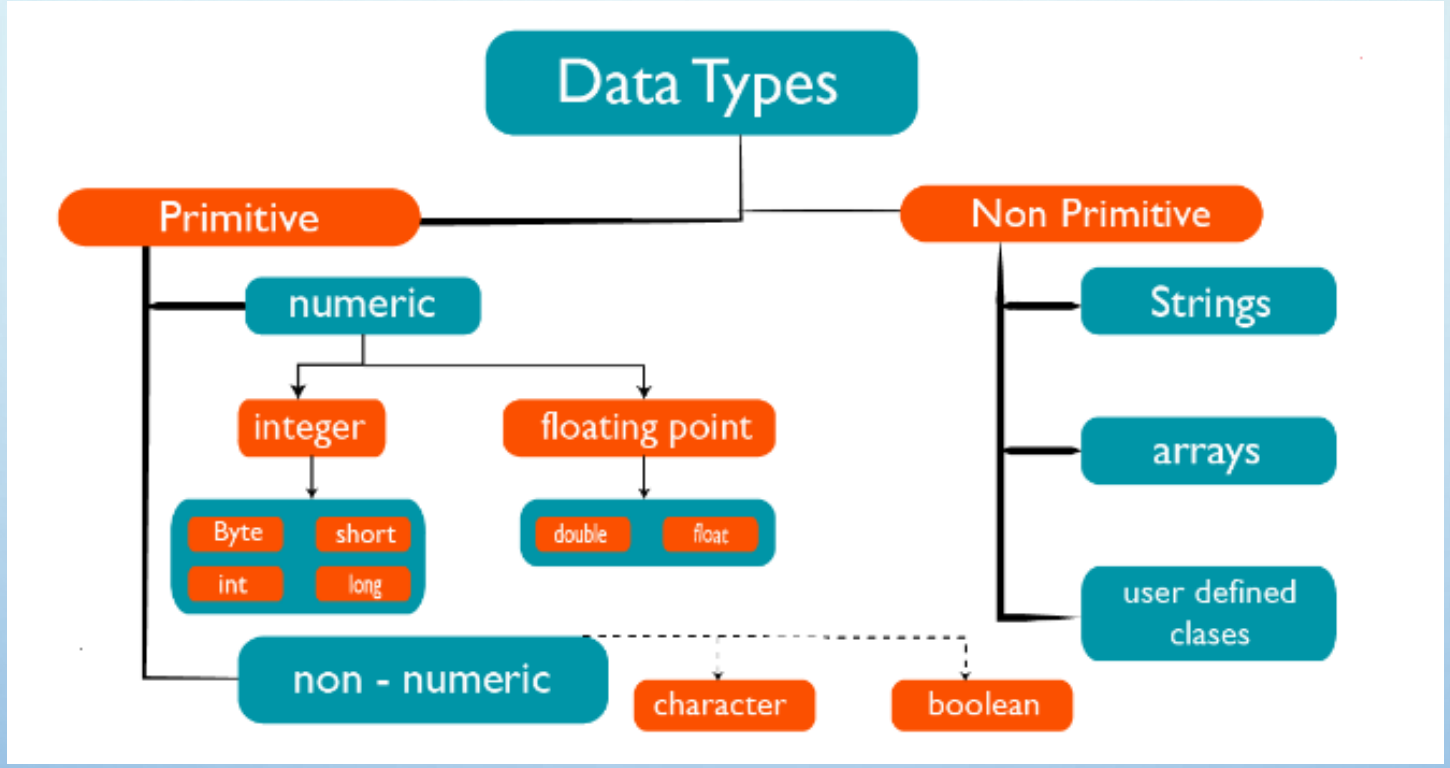
JAVA DATA TYPES

By Ayala Berkovich



By Ayala Berkovich

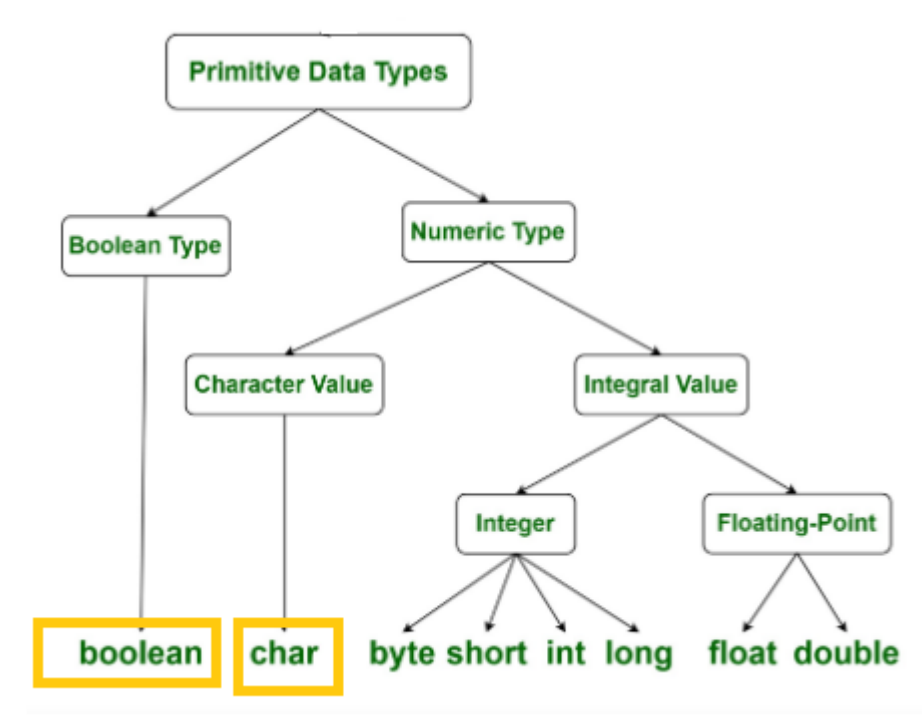
CAN JAVA BE SAID TO BE THE COMPLETE OBJECT-ORIENTED PROGRAMMING LANGUAGE?

No! In JAVA we use data types like int, float, double etc which are not object oriented, and of course is what opposite of OOP is.

By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES

PRIMITIVES DEFINED IN JAVA ARE **INT, BYTE, SHORT, LONG, FLOAT, DOUBLE, BOOLEAN AND CHAR**.



Boolean data type represents only **one bit** of information- **true or false**

Default Value: false

Char data type is a single **16-bit** Unicode character

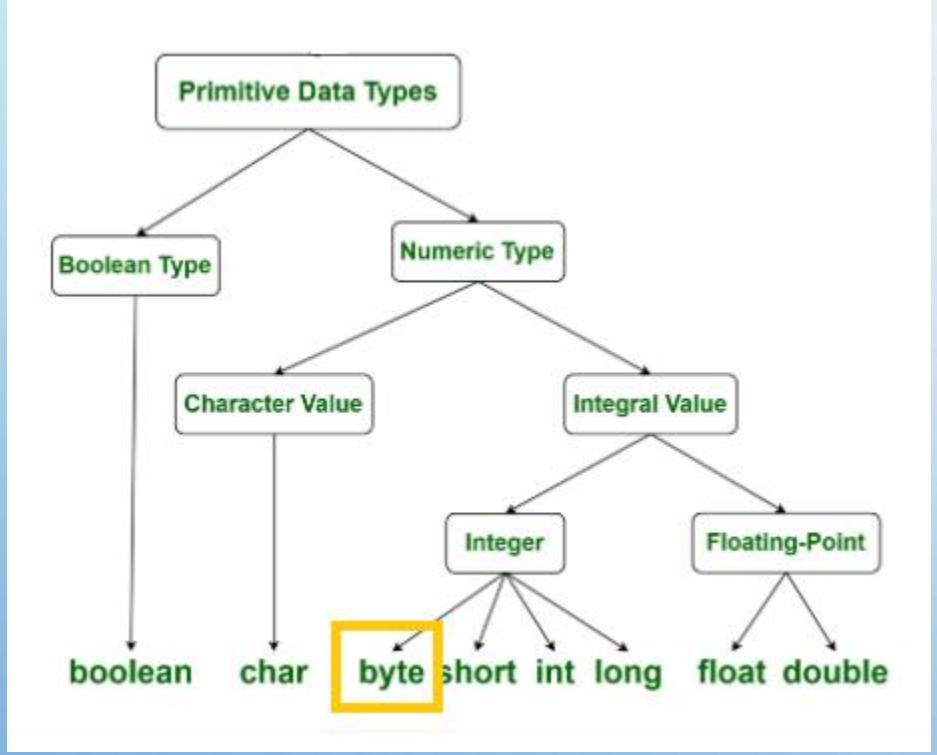
Default value: *'\u0000'*

'\u0000' what it is in Java?

equivalent to NULL

By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



Byte data type is an **8-bit** signed two's complement integer. The byte data type is useful for saving memory in large arrays.

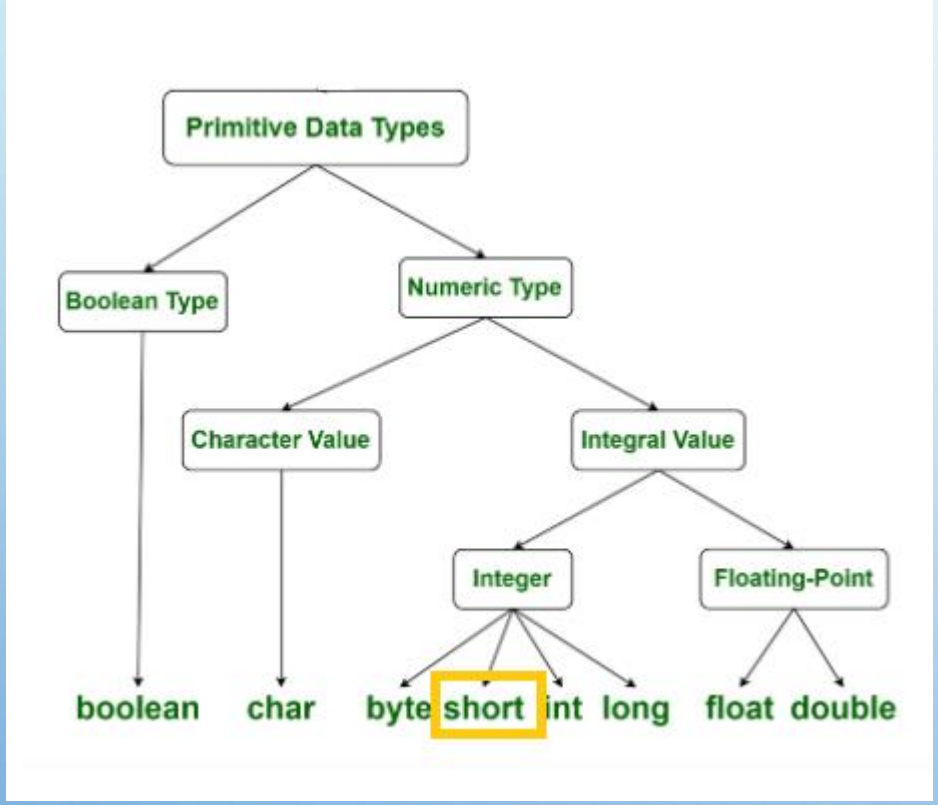
Default Value: 0

Eight-bit signed integers

Decimal value	Two's-complement representation
0	0000 0000
1	0000 0001
2	0000 0010
126	0111 1110
127	0111 1111
−128	1000 0000
−127	1000 0001
−126	1000 0010
−2	1111 1110
−1	1111 1111

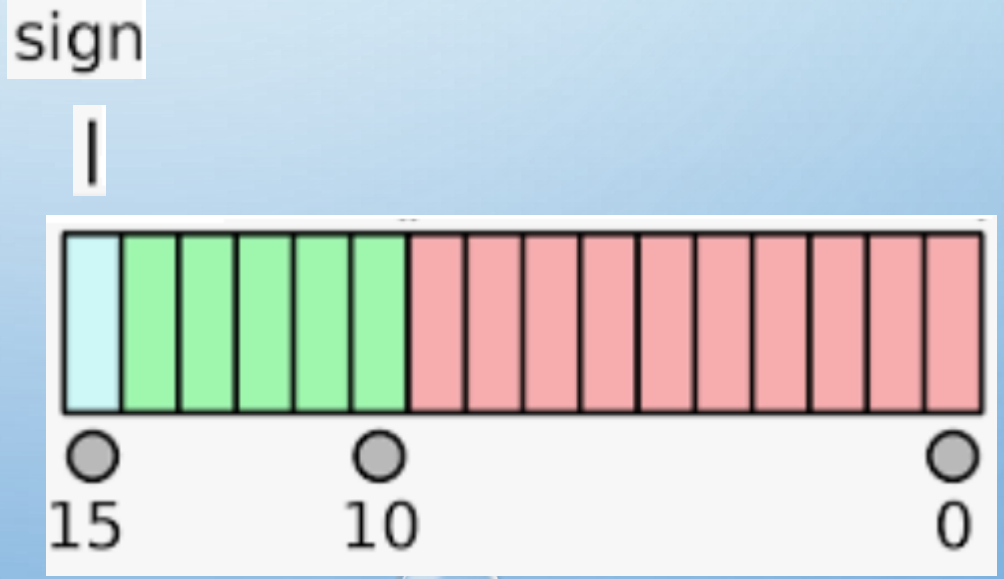
By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



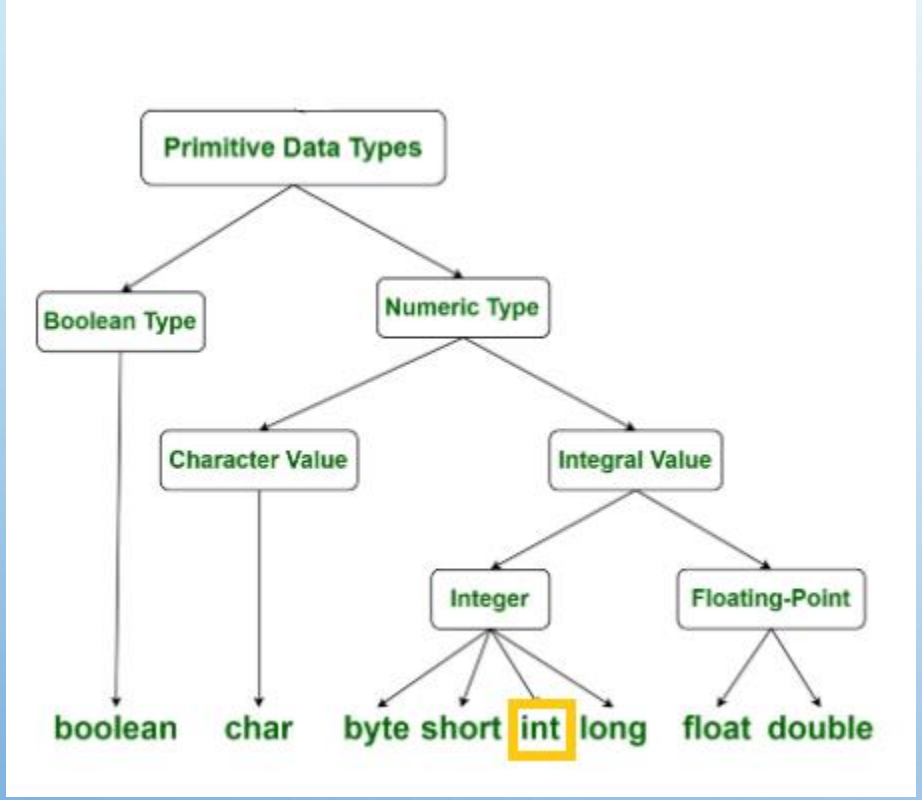
Short data type is a **16-bit** signed two's complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters …

Default Value: 0



By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



Int data type is **32-bit** signed two's complement integer

Default Value: 0

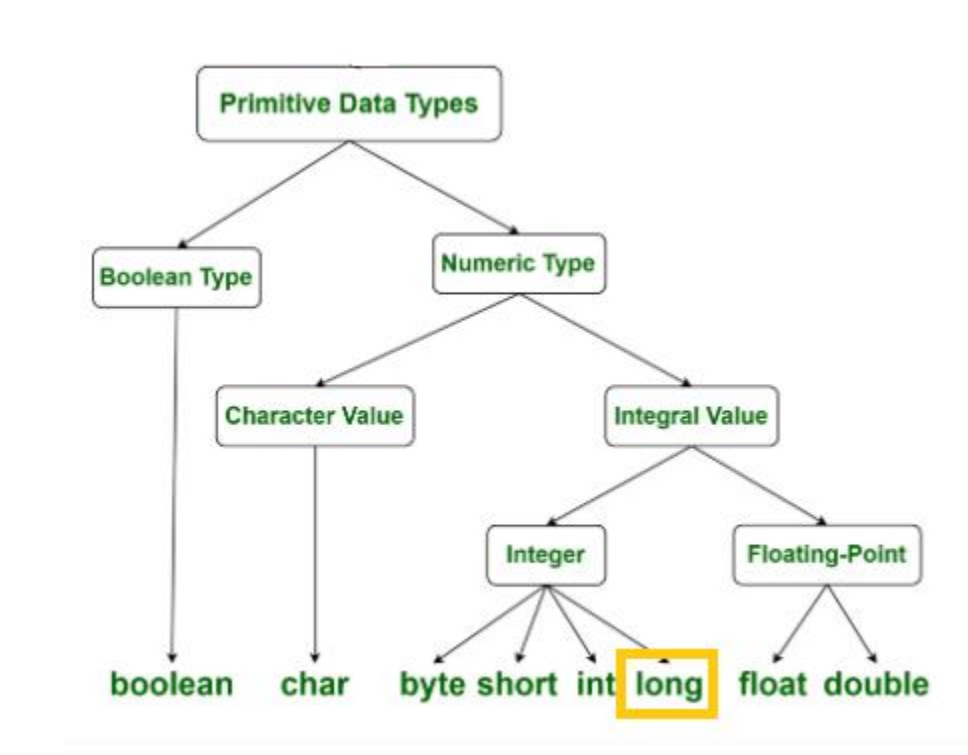
The last bit is used to distinguish positive and negative numbers.
If the last bit is NOT set, then the number is positive. Therefore, the maximal positive number is 0x7FFFFFFF = (1<<31)-1=2147483647 (the last bit is not set).

The minimal number in two's complement notation is 0x80000000 = -2147483648

1 10011001 11010110111100101000000

By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



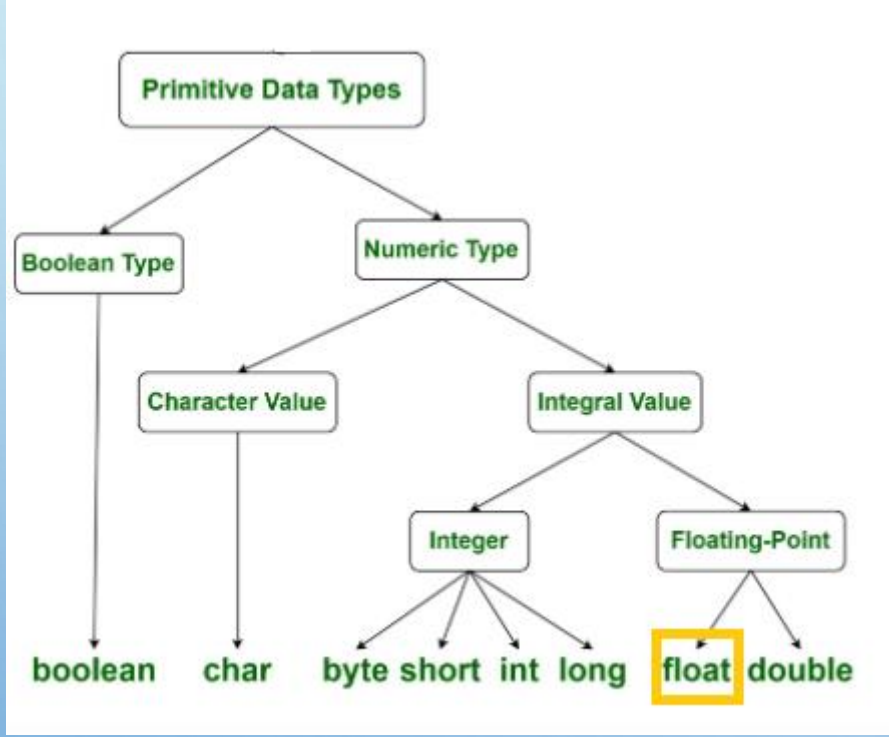
Long data type is **a 64-bit** two's complement integer and is useful for those occasions where an int type is not large enough to hold the desired value

Default Value: 0

1 10000011001 1101011011110010011111111111111111111111111111111111

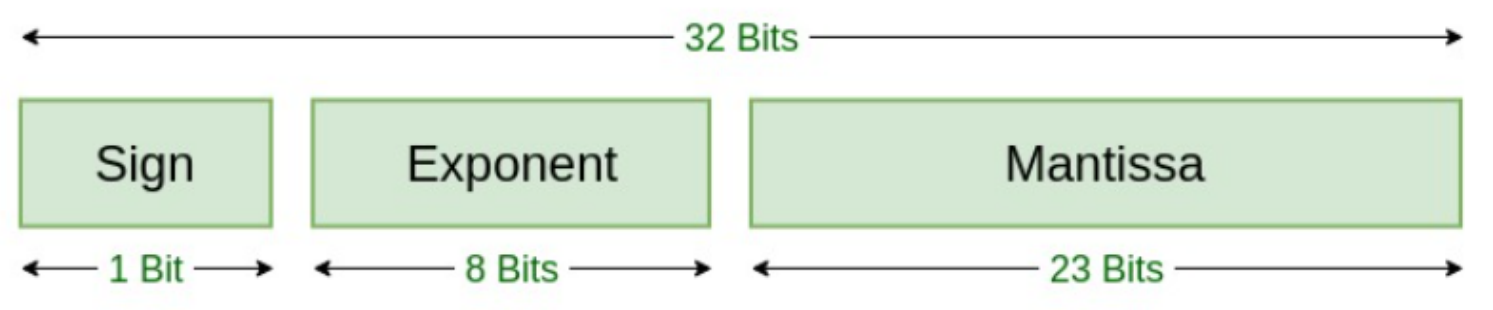
By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



Float data type is a single-precision **32-bit** IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers

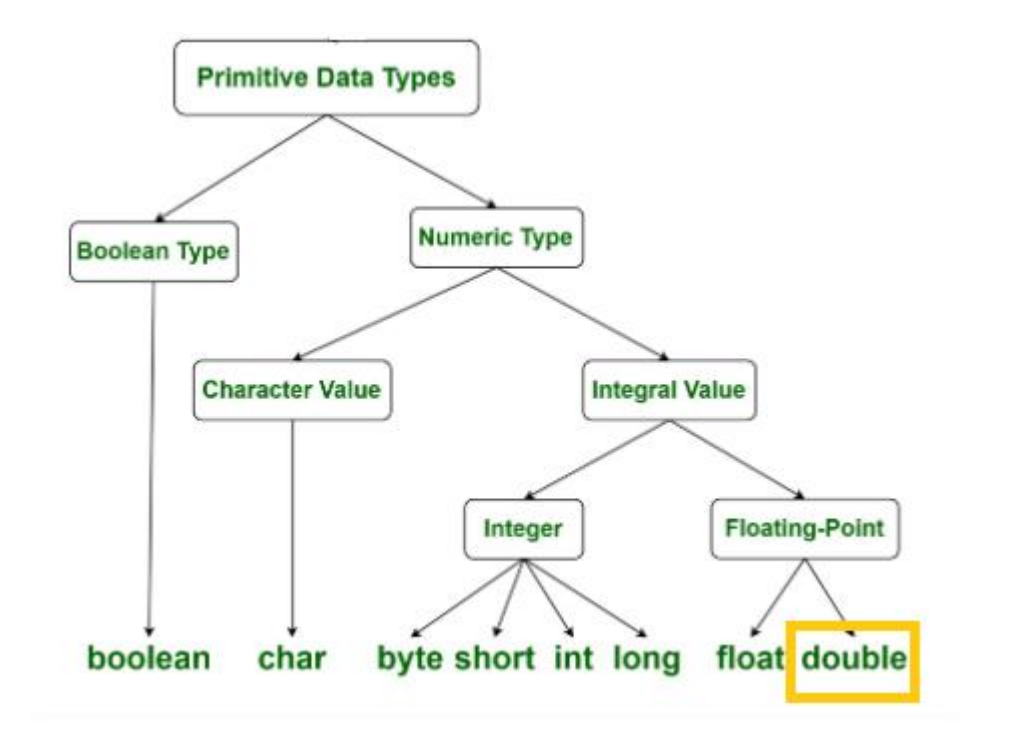
Default Value: 0.0



In number 2.3010 - 2 is exponent
- 3010 is mantissa

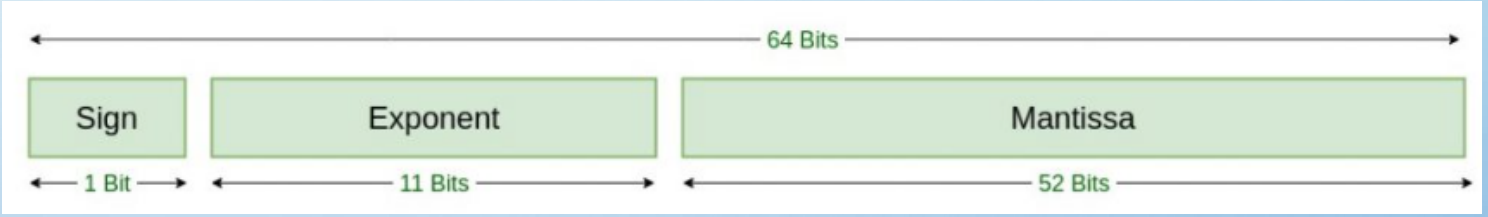
By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES



Double data type is a double-precision **64-bit** IEEE 754 floating-point

Default Value: 0.0



By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES

BUT PRIMITIVE DATA TYPES IT IS VERY SIMPLE … WHY BACK ON THIS NOW?

Many programmers thinks that it is very simple….but in code we can see for example:

- **Integer overflow** and **underflow** cases -
  happen when we assign a value that is out of range of the declared data type of the variable

  Integer of type ***int*** in Java can be negative or positive, which means with its 32 bits, we can assign values between -2^{31} **(-2147483648) and** 2^{31}**-1 (2147483647**).

  class *Integer* defines two constants that hold these values: ***Integer.MIN_VALUE*** and ***Integer.MAX_VALUE***.

By Ayala Berkovich

PRIMITIVE JAVA DATA TYPES

What will happen if we define a variable *m* of type *int* and attempt to assign a value that's too big (e.g., *2147483647 = MAX_VALUE + 1)?*

```
int value = Integer.MAX_VALUE-1;
for(int i = 0; i < 4; i++, value++) {
    System.out.println(value);
}
```

```
2147483646
2147483647
-2147483648
```

If it overflows, **it goes back to the minimum value and** continues from there. If it underflows, it goes back to the **maximum value** and continues from there..... **This behavior is called integer-wraparound**.

Java does not throw an exception when an overflow occurs!!!!

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

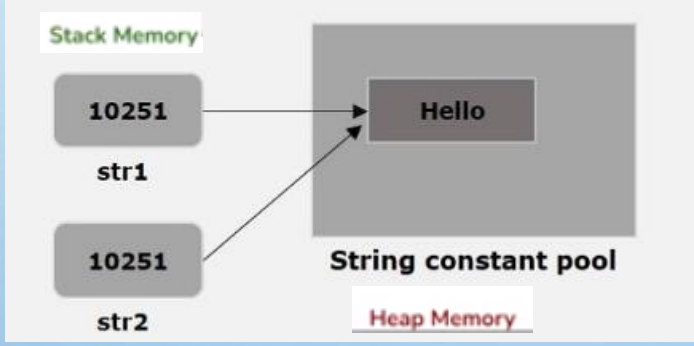
STRING IS A COLLECTION OF CHARACTERS

We have two options to create string...

First:

```
String str_name = "str_value";
```

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



In this way, the new string object will be created in the heap

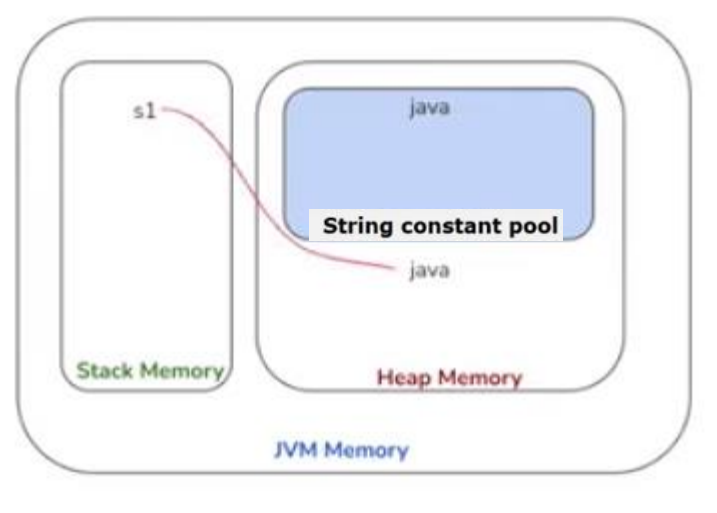
JVM creates a String object with the given value in a ***String constant pool***(memory allocated in heap for future use)

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

Second by New keyword:

```
String s1 = new String("java")
```

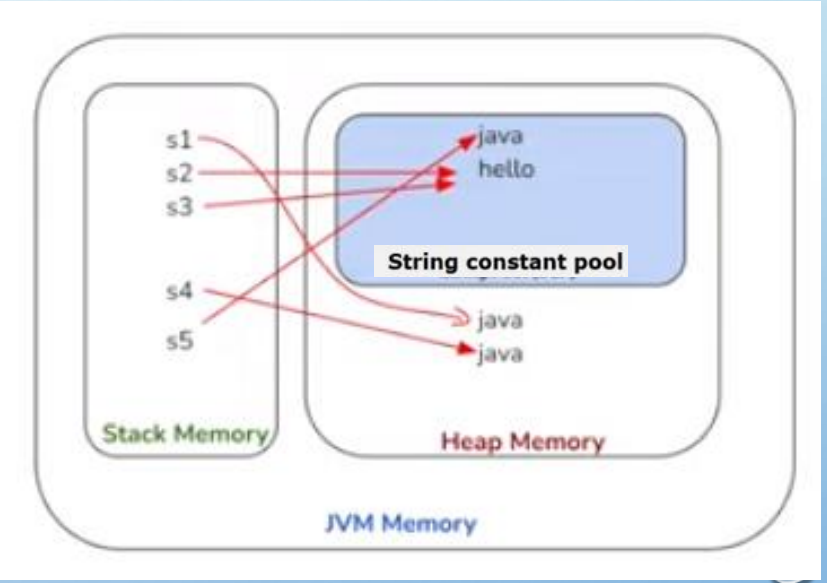


By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

How many objects will be created in case of:

String s1 = new String("java")	2
String s2 = "hello"	1
String s3 = "hello"	0
String s4 = new String("java")	1
String s5 = "java"	0



By Ayala Berkovich

Heap_Inspection

String variables are immutable - in other words, once a string variable is assigned, its value cannot be changed or removed. Thus, these strings may remain around in memory, possibly in multiple locations, for an indefinite period of time until the garbage collector happens to remove it. Sensitive data, such as passwords, will remain exposed in memory as plaintext with no control over their lifetime.

While it may still be possible to retrieve data from memory, even if it uses a mutable container that is cleared, or retrieve a decryption key and decrypt sensitive data from memory - layering sensitive data with these types of protection would significantly increase the required effort to do so. By setting a high bar for retrieving sensitive data from memory, and reducing the amount and exposure of sensitive data in memory, an adversary is significantly less likely to succeed in obtaining valuable data.

By Ayala Berkovich

General Recommendations

- Do not store sensitive data, such as passwords or encryption keys, in memory in plain-text, even for a short period of time.
- Prefer to use specialized classes that store encrypted data in memory to ensure it cannot be trivially retrieved from memory.
- When required to use sensitive data in its raw form, temporarily store it in mutable data types, such as byte arrays, to reduce readability from memory, and then promptly zeroize the memory locations, to reduce exposure duration of this data while in memory.
- Ensure that memory dumps are not exchanged with untrusted parties, as even by ensuring all of the above - it may still be possible to reverse-engineer encrypted containers, or retrieve bytes of sensitive data from memory and rebuild it.
- In Java, do not store passwords in immutable strings - prefer using an encrypted memory object, such as SealedObject.

By Ayala Berkovich

Java

Plaintext Password in Immutable String

```
class Heap_Inspection
{
  private String password;

  public void setPassword(String password)
  {
        this.password = password;
  }
}
```

Password Protected in Memory

```
class Heap_Inspection_Fixed
{
  private SealedObject password;

  public void setPassword(Character[] input)
  {
        Key key = getKeyFromConfiguration();
                Cipher c = Cipher.getInstance(CIPHER_NAME);
                c.init(Cipher.ENCRYPT_MODE, key);
                List<Character> characterList = Arrays.asList(input);
                password = new SealedObject((Serializable) characterList, c);
                Arrays.fill(input, '\0'); // Zero out input. Will also overwrite the values in characterList by reference.
```

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

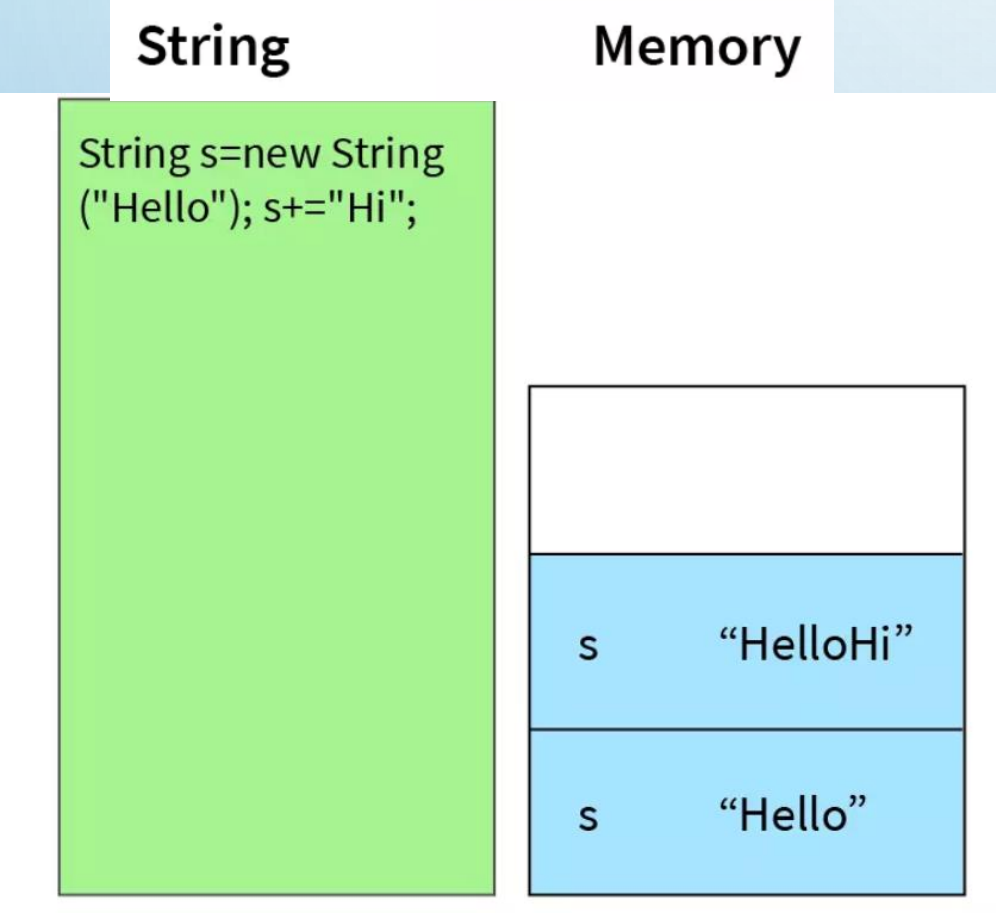
!Remember? No save **sensitive** data in String just after encryption....

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

STRING IT IS IMMUTABLE CLASS

IMMUTABLE CLASS IN JAVA MEANS THAT ONCE AN OBJECT IS CREATED, WE CANNOT CHANGE ITS CONTENT. IN JAVA, **ALL THE WRAPPER CLASSES (LIKE INTEGER, BOOLEAN, BYTE, SHORT) AND STRING CLASS IS IMMUTABLE**.

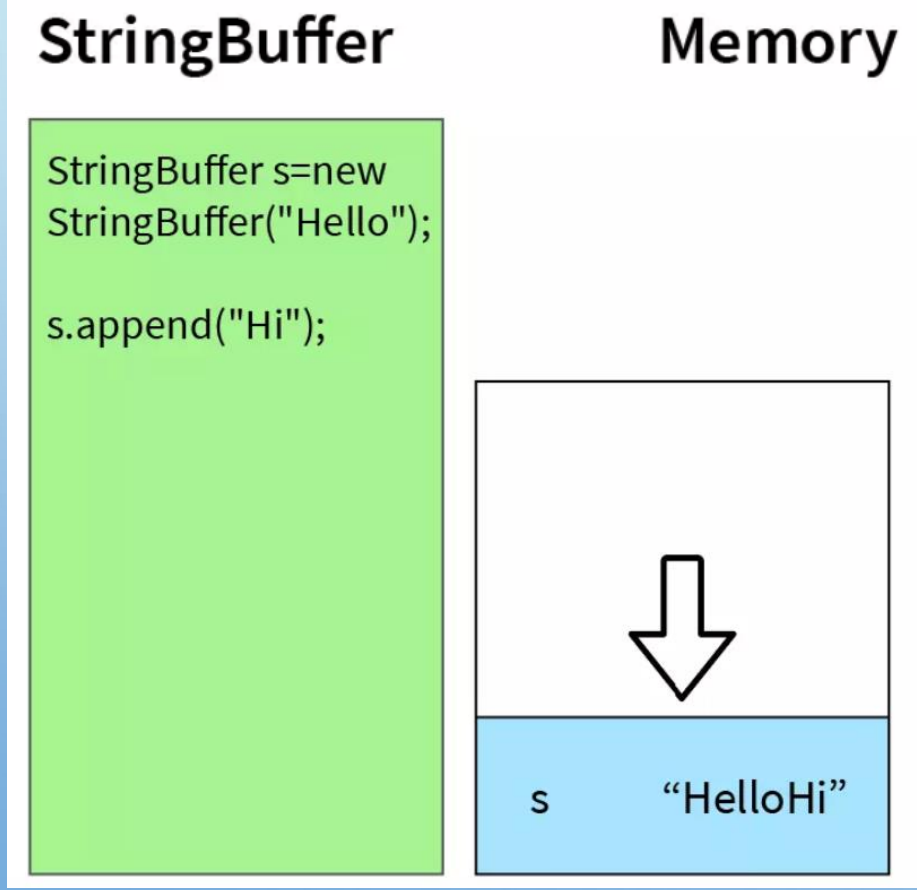


Operations with string data type:

Substring() – Returns the substring of the string.
Concat() – Concatenates the string.
Length () – Returns the length of the string
valueOf – Convert to string from other data types

NON-PRIMITIVE JAVA DATA TYPES

By Ayala Berkovich



STRINGBUFFER - IS USED TO CREATE MUTABLE (MODIFIABLE) STRING. IT WILL AUTOMATICALLY GROW.

StringBuffer() constractor : creates an empty string buffer with the initial capacity of **16**(capacity -how much memory the string allocated to hold its contents, for example: string with capacity 8 could hold 8 characters.)

Functions:

- **append()** method concatenates the given argument with this string.
- **insert()** method inserts the given string with this string at the given position.
- **replace()** method replaces the given string from the specified beginIndex and endIndex-1.
- **delete()** method of StringBuffer class deletes the string from the specified beginIndex to endIndex-1.
- **reverse()** method of StringBuilder class reverses the current string.
- **capacity()** method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

STRINGBUILDER - IS USED TO CREATE MUTABLE (MODIFIABLE) STRING.

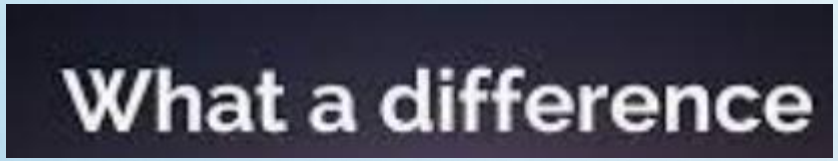
STRINGBUILDER() CONSTRACTOR : CREATES AN EMPTY STRING BUFFER WITH THE **INITIAL CAPACITY OF 16.**

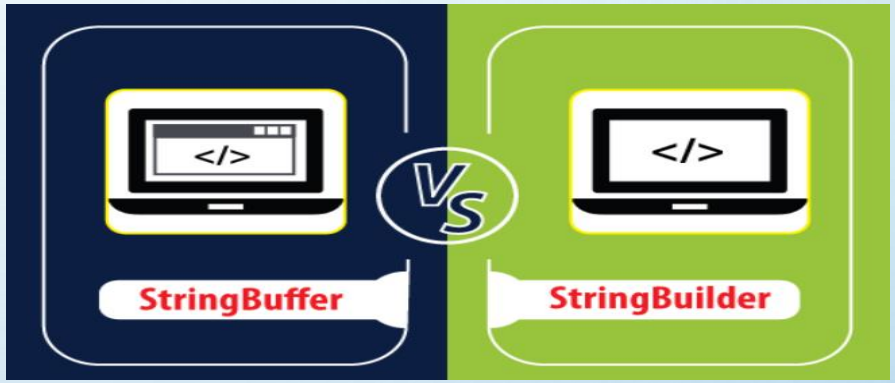
Functions:

- **append()** method concatenates the given argument with this string.
- **insert()** method inserts the given string with this string at the given position.
- **replace()** method replaces the given string from the specified beginIndex and endIndex-1.
- **delete()** method of StringBuilder class deletes the string from the specified beginIndex to endIndex-1.
- **reverse()** method of StringBuilder class reverses the current string.
- **capacity()** method of StringBuilder class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.

By Ayala Berkovich

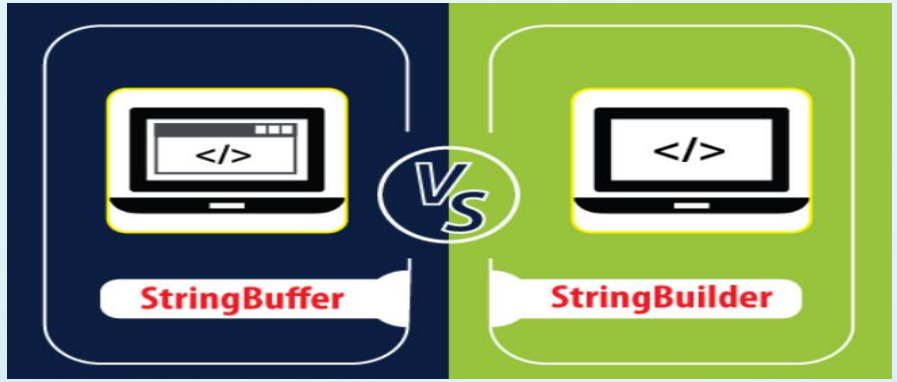
NON-PRIMITIVE JAVA DATA TYPES





No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

By Ayala Berkovich




```java
public class ConcatTest{
  public static void main(String[] args){
    long startTime = System.currentTimeMillis();
    StringBuffer sb = new StringBuffer("Java");
    for (int i=0; i<10000; i++){
      sb.append("Tpoint");
    }
    System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
    startTime = System.currentTimeMillis();
    StringBuilder sb2 = new StringBuilder("Java");
    for (int i=0; i<10000; i++){
      sb2.append("Tpoint");
    }
    System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
  }
}
```

Output:

```
Time taken by StringBuffer: 16ms
Time taken by StringBuilder: 0ms
```

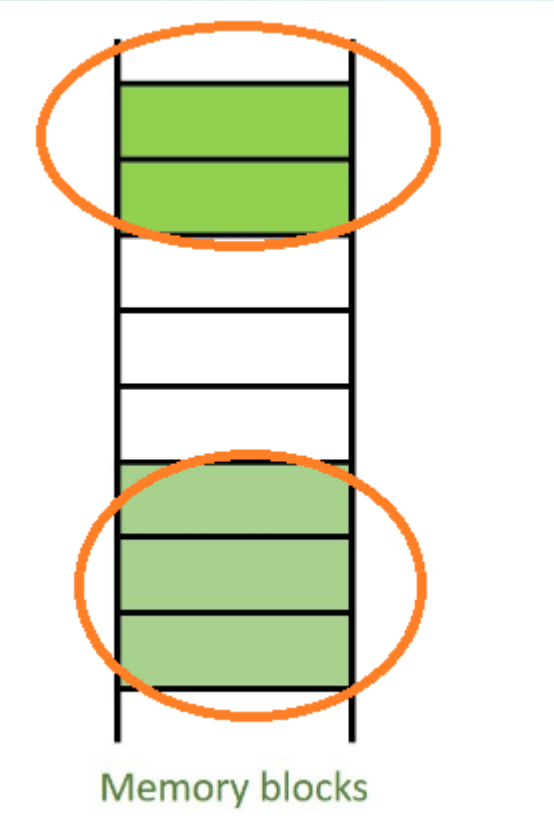
StringBuilder is *more efficient* than StringBuffer.

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

ARRAY IS AN OBJECT WHICH CONTAINS ELEMENTS OF A SIMILAR DATA TYPE

In Java, all arrays are dynamically allocated and stored in contiguous memory.
Consecutive blocks of memory allocated to user processes are called contiguous memory.
For example, if a user process needs some x bytes of contiguous memory, then all the x bytes will reside in one place in the memory that is defined by a range of memory addresses: 0x0000 to 0x00FF.
Single contiguous section/part of memory is allocated to a process.
The size of the array cannot be altered(once initialized).




```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

*The elements in the array allocated by new will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types).*

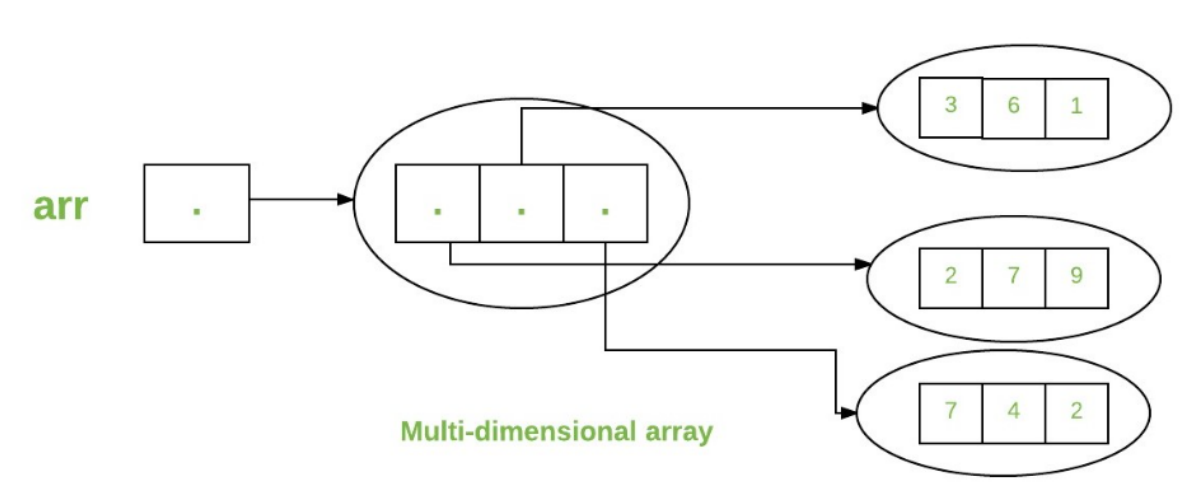
What happens if we try to access elements outside the array size?

Will throws **ArrayIndexOutOfBoundsException**

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

MULTIDIMENSIONAL ARRAYS - EACH ELEMENT OF THE ARRAY HOLDING THE REFERENCE OF OTHER ARRAYS.



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

```java
public class multiDimensional {
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][]
            = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

        // printing 2D array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(arr[i][j] + " ");

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

Output

```
2 7 9
3 6 1
7 4 2
```

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

USER DEFINED CLASSES DATA TYPES IN JAVA - DATA TYPES CREATED BY THE DEVELOPERS.

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

By Ayala Berkovich

NON-PRIMITIVE JAVA DATA TYPES

User defined classes data

- Create base class Person with firstName, lastName, address and role.
- Create Class Student;
- Create Class Teacher;

With printRole() function that will output role of the person…

- Run and check that you defined it correct …


THANK YOU