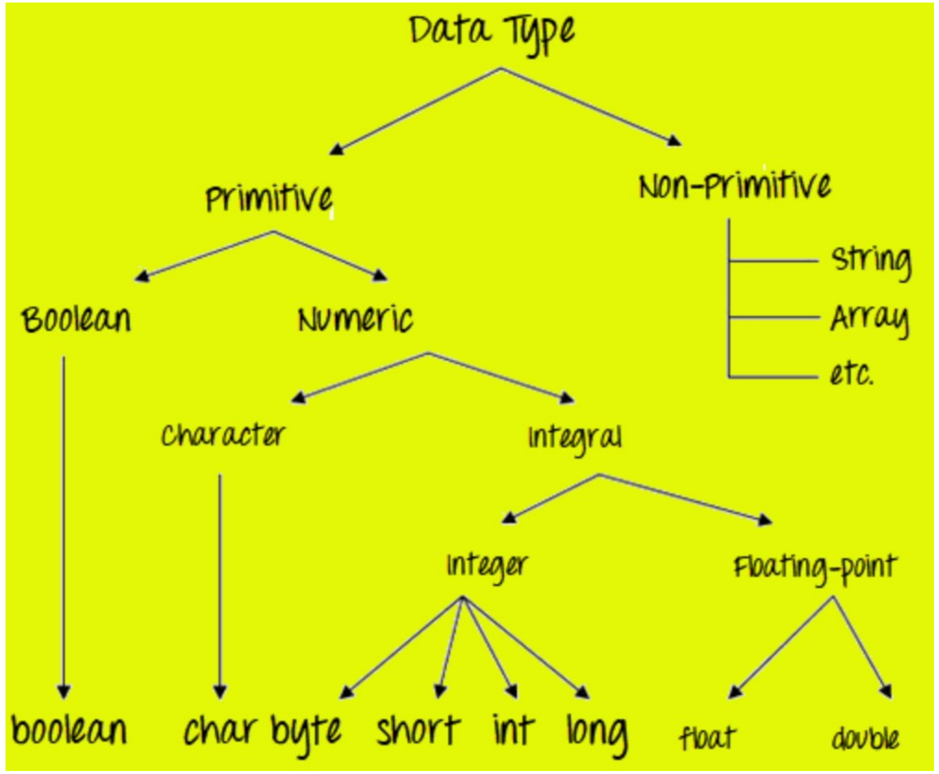
Advanced Programming Techniques in Java

COSI 12B

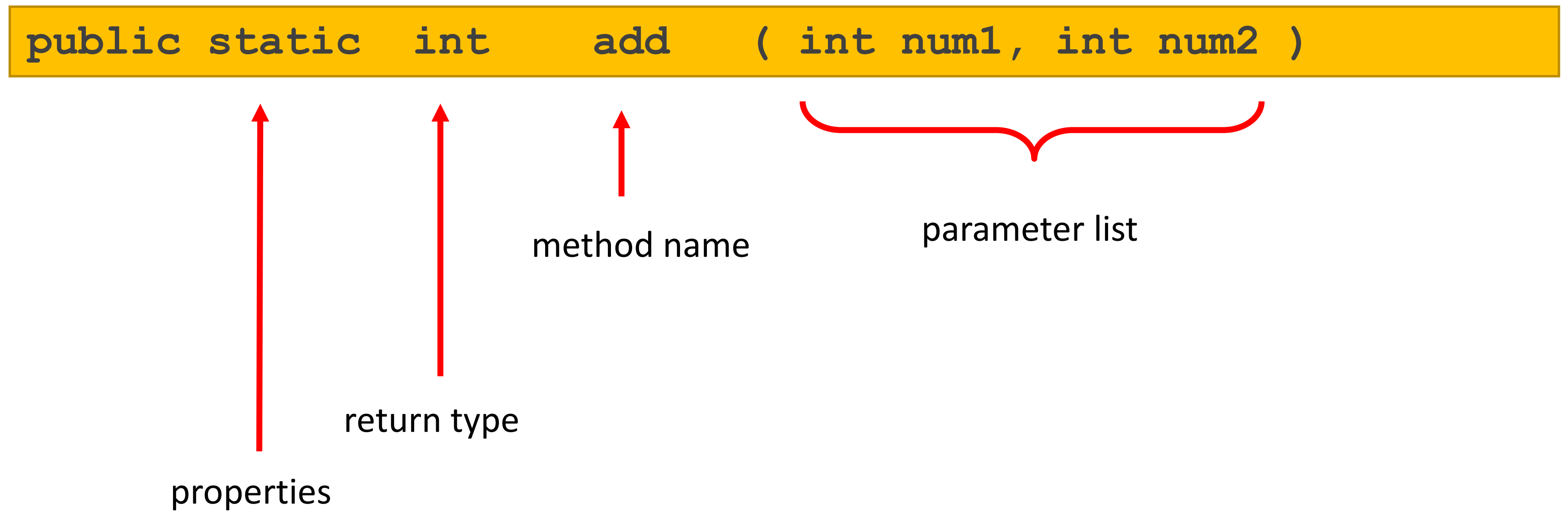
Review: Data Types

- Data types classify the different values to be stored in the variable
- In Java there are two types of data types:
  - Primitive Data Types
  - Non-primitive Data Types



Review: Methods

- A method declaration begins with a *method header*



```
public static  int    add   ( int num1, int num2 )
```

- The parameter list specifies the type and name of each parameter
- The name of a parameter in the method declaration is called a *formal argument*
- **static** indicates a *static* or an *object/instance* method
- A method that is not static, is an instance method

Review: Parametrization

- A *parameter* is a special type of variable that allows us to pass information into a method
- A method can accept multiple parameters (separated by ,) including none
- Each time a method is called, the *actual parameters* in the invocation are copied into the formal

Declaration syntax

```
public static  int    add   ( int num1, int num2 )
```

formal parameters

Call syntax

```
add (5, 9);
```

actual parameters

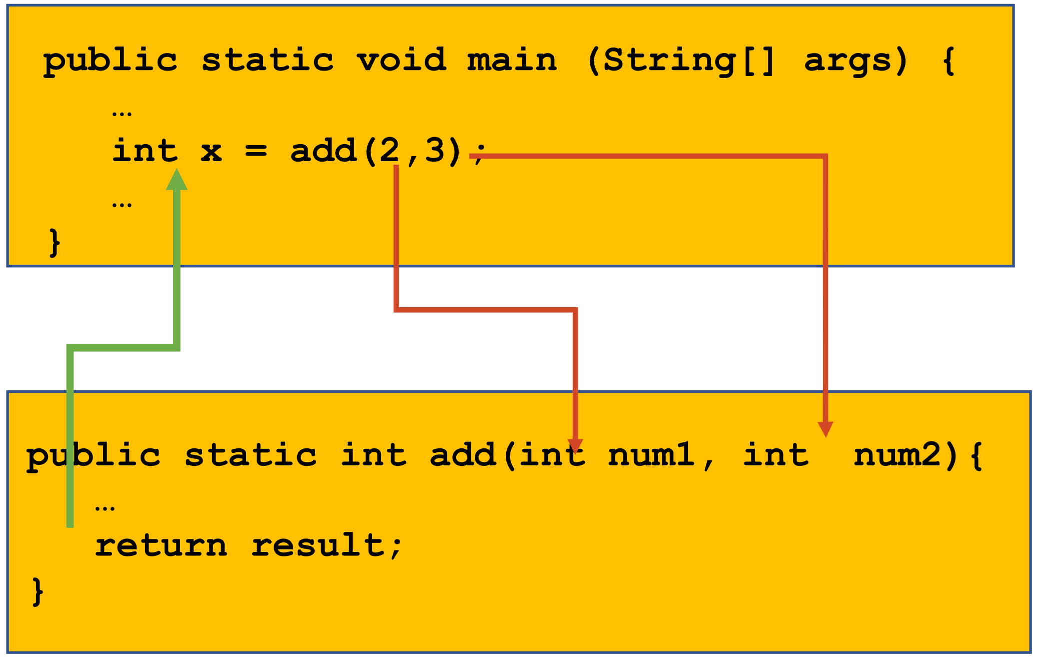
To summarize …

```
public class AddingTwoNumbers{

    public static void main (String[] args){
        int x = add (2, 3);    // method call
        System.out.println(x);
    }

    public static int add (int num1, int num2){
            int result = 0;
            result = num1 + num2;
            return result;
    }

}
```



```
public static void main (String[] args) {
   …
   int x = add(2,3);
   …
}

public static int add(int num1, int  num2){
   …
   return result;
}
```

`main` Method

- The `main` method is a special method with a specific header from which the execution of a program starts

```
public static void main(String args[]) {
    // method body
}
```

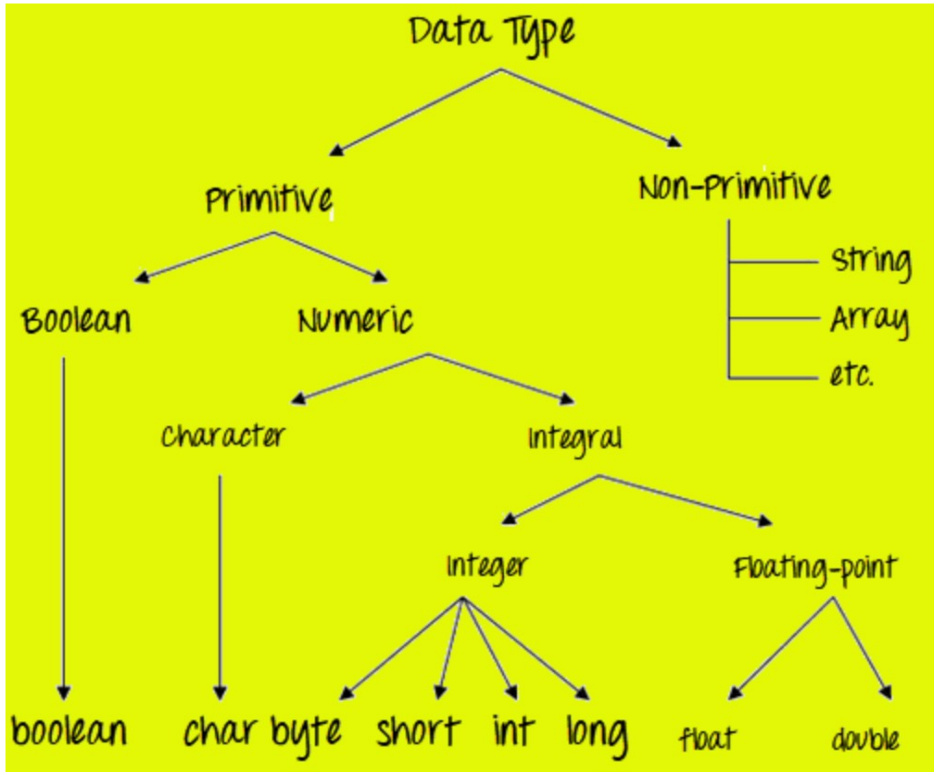
Class objectives

- Strings
- Primitive and Reference (Non-Primitive) types
- Randomization

Strings

Data Types

- Data types classify the different values to be stored in the variable
- In Java there are two types of data types:
  - Primitive Data Types
  - Non-primitive Data Types



Strings

- `"Hello, world!"` or `"Enter a number: "` are *strings*
- Java supplies a class called `String` used to create and process strings

- A *string* is an object storing a sequence of characters

- String objects have
  - Fields (or data values): the characters in the string
  - Methods (or operations): get the length of the string, get a substring, etc.

- Strings in Java are immutable, which means that once they are constructed, their value can never change

Strings

- `"Hello, world!"` or `"Enter a number: "` are *strings*
- Java supplies a class called `String` used to create and process strings
  - We'll first learn how to use objects and later how to create them

- A *string* is an object storing a sequence of characters

- Objects have
  - Fields (or data values). For strings the fields are the characters in the string.
  - Methods (or operations). For strings some of the operations are get the length of the string, get a substring, etc.

Strings

- Unlike most other objects, a `String` does not have to be instantiated (created) using the keyword `new`
- You can simply declare variable of type `String` and assign a value to it (this is special syntax that works only for strings)

```
String s1 = "hello";
String s2 = "there";
```

- When declaring a string you can have a value or an expression

```
String combined = s1 + " " + s2;
```

- Remember the **+** symbol concatenates strings to form a larger string `"hello there"`

Strings

- The `String` class provides many methods that can be used to manipulate strings.
  - Many of the methods return a value, such as an integer or a new `String` object
- Once an object has been instantiated, we can use the dot operator to invoke its methods

- Given:

```
String s1 = "hello";
String s2 = "how are you?";
```

- To know how many characters are in `s1` and `s2` you can use the `length()` method

```
System.out.println("Length of s1: " + s1.length());
System.out.println("Length of s2: " + s2.length());
```

Strings

- The **`length()`** method returns the length of the string (i.e., how many characters are in the string)

- How to get the individual characters by themselves?

Strings

- The characters of a string are numbered with 0-based **indexes**

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

  - First character's index : 0
  - Last character's index : 1 less than the string's length
  - The individual characters are values of type `char`

`charAt(index)` method returns the character at the index location in the string

```
name.charAt(0)         'R'
name.charAt(3)         'K'
```

Strings

- How to get a substring?

  `String s2 = "How are you?"` (e.g. the substring is `"are"`)

index	0	1	2	3	4	5	6	7	8	9	10	11
character	H	o	w		a	r	e		y	o	u	?

- The method **`substring(start, end)`** returns a new string having the same characters as the substring that begins at index `start` through, <u>but not including</u>, `end`

```
s2.substring(8,12);        "you?"
s2.substring(4,7);         "are"
```

Strings

Method name	Description
`charAt(`**`index`**`)`	Returns the character at the index location in the string
`length()`	Returns the number of characters in this string
`substring(`**`index1, index2`**`)` or `substring(`**`index1`**`)`	Returns the characters in this string from **`index1`** (inclusive) to **`index2`** (exclusive); if **`index2`** is omitted, grabs till end of string
`toLowerCase()`	Returns a new string with all lowercase letters
`toUpperCase()`	Returns a new string with all uppercase letters
…	

String example

```
public class Test {
   public static void main(String[] args) {
      String s1 = "hello";
      String s2 = "class";
      String s3 = "soon we’ll have the first holiday!";

      System.out.println(s1 + " " + s2 + " " + s3);
      System.out.println();

      //Use of the method length()
      String s4 = s1 + " " + s2 + " " + s3;
      int strLen = s4.length ();
      System.out.println("The length of the string s4 is: " + strLen);
      System.out.println();

      //Use of the method charAt(index)
      char ch = s4.charAt (3);
      System.out.println("The character in string s4 at location 3 is: " + ch);
      System.out.println();

      //Use of the method toUpperCase()
      String newString = s4.toUpperCase();
      System.out.println(newString);
   }
}
```

`char` vs. `String`

- `char` is a primitive type representing a single character. E.g., `'h'`
- A string is an object. E.g., `"h"`

```
String s = "h";
s = s.toUpperCase();          // "H"
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- A `char` is a primitive data type; you can't call methods on it

Comparing `char` values

- You can compare `char` values with `==`, `!=`, and other operators

```
String word = console.next();
char last = word.charAt(word.length() - 1);
if (last == 's') {
      System.out.println(word + " is plural.");
}
```

```
for (char c = 'a'; c <= 'z'; c++) {
      System.out.print(c);
}
```

`char` vs. `int`

- Each `char` is mapped to an integer value, called an ASCII value

  `'A'` is 65 `'B'` is 66 `' '` is 32

  `'a'` is 97 `'b'` is 98 `'*'` is 42

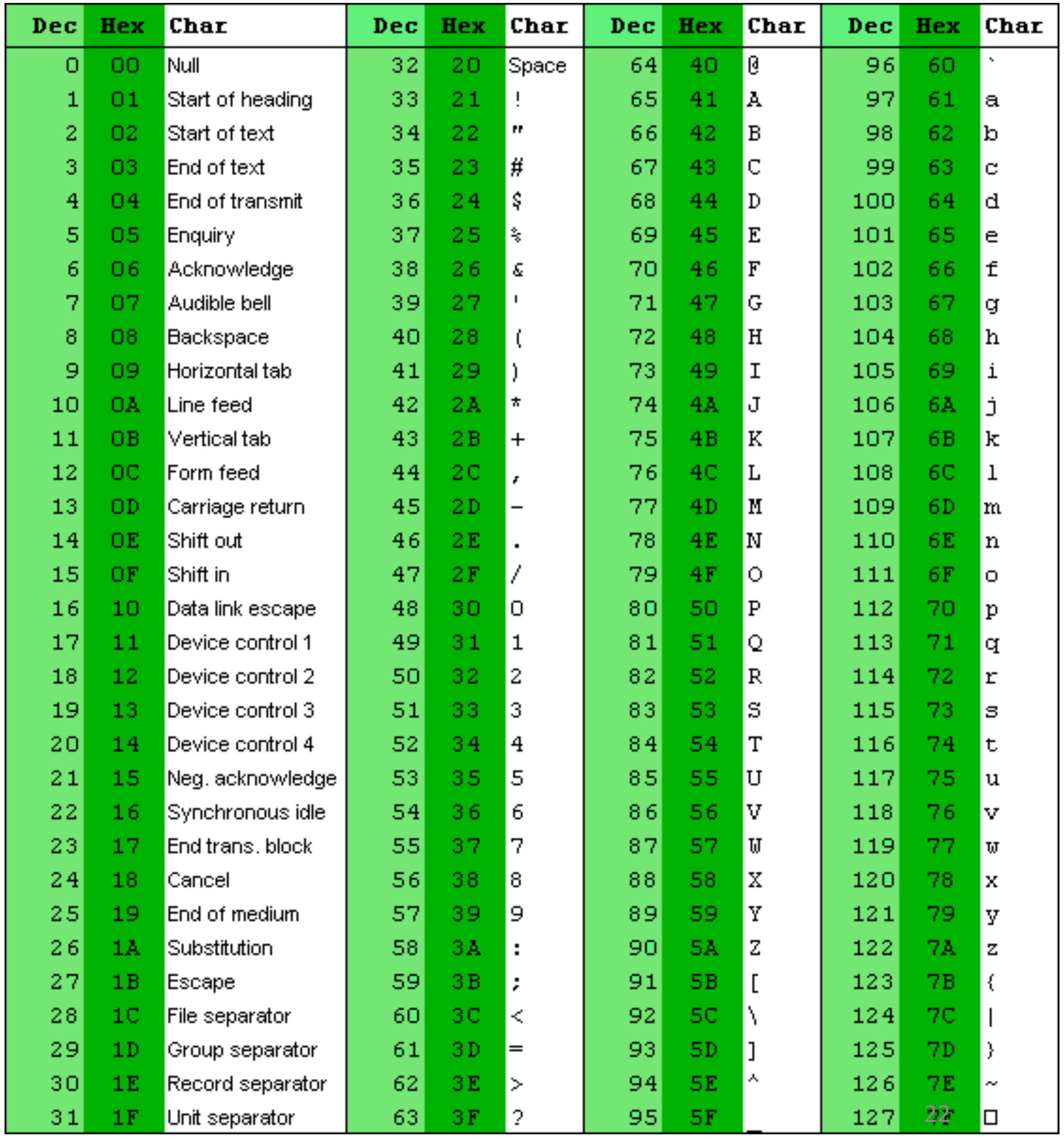
- Mixing a `char` and an `int` causes automatic conversion to `int`

  `'a'` + 10 is 107

- To convert an `int` into the equivalent `char`, type-cast it

  `(char) ('a' + 2)` is `'c'`

ASCII table



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Comparing Strings

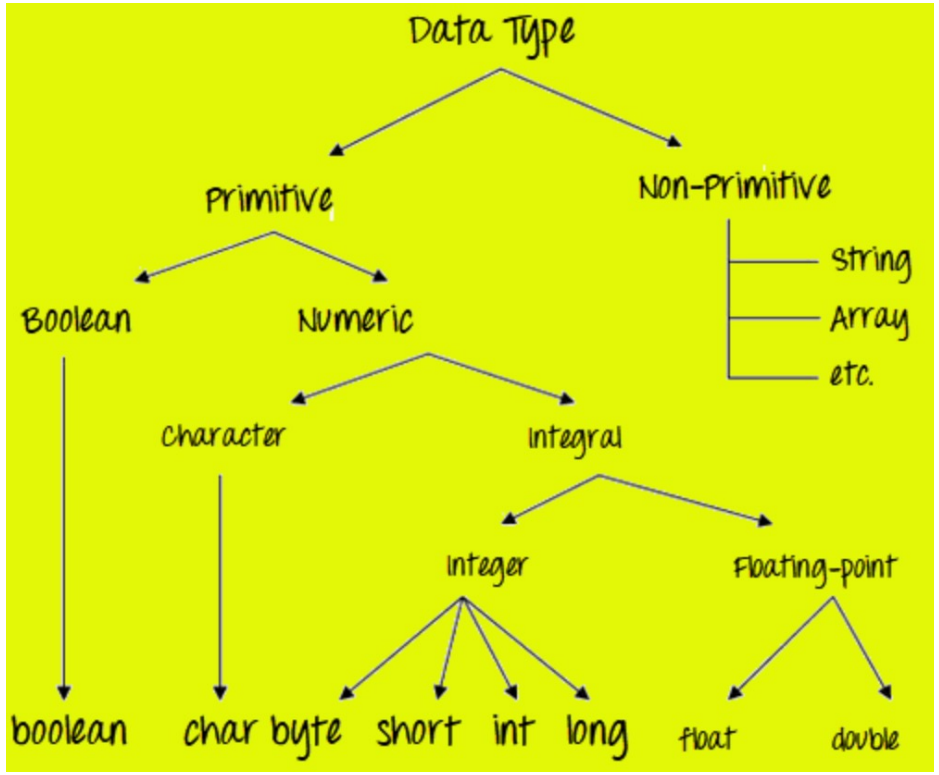
- Relational operators such as < and == fail on objects

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
   System.out.println("I love you, you love me,");
   System.out.println("We're a happy family!");
}
```

- This code will compile, but it will not print.

Data Types

- Data types classify the different values to be stored in the variable
- In Java there are two types of data types:
  - Primitive Data Types
  - Non-primitive Data Types



The `equals` method

- Objects are compared using a method named `equals`

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

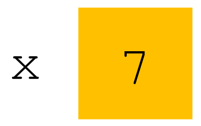
- This is a method that returns a value of type `boolean`, the type used in logical tests

Primitive and Reference Type

Primitives, Objects, and References

- Some types of data are stored inside their variables

```
int x = 7;
```

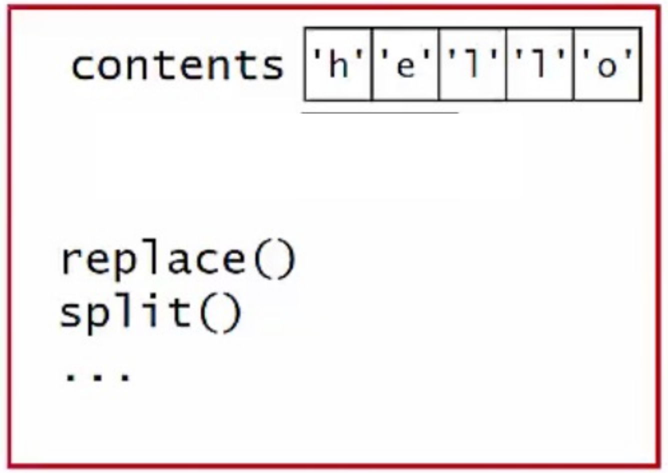


- These data types are known as primitive types:
  - `int`
  - `long`
  - `double`
  - `boolean`
  - `char`

Object vs. Primitive

- If something is not primitive, it is an object
- An object is a construct that groups together:
  - one or more data values (the object's *attributes* or *fields*)
  - one or more *methods*
- Every object is referred as an **instance** of a class

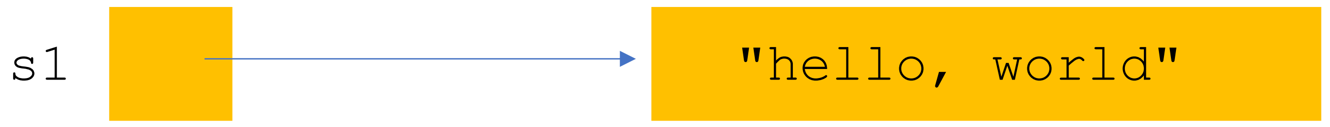
String object



Reference Types

- Objects are stored as reference
  - The object is stored outside the variable
  - The variable stores a reference (memory address) to the object

```
String s1 = "hello, world";
```

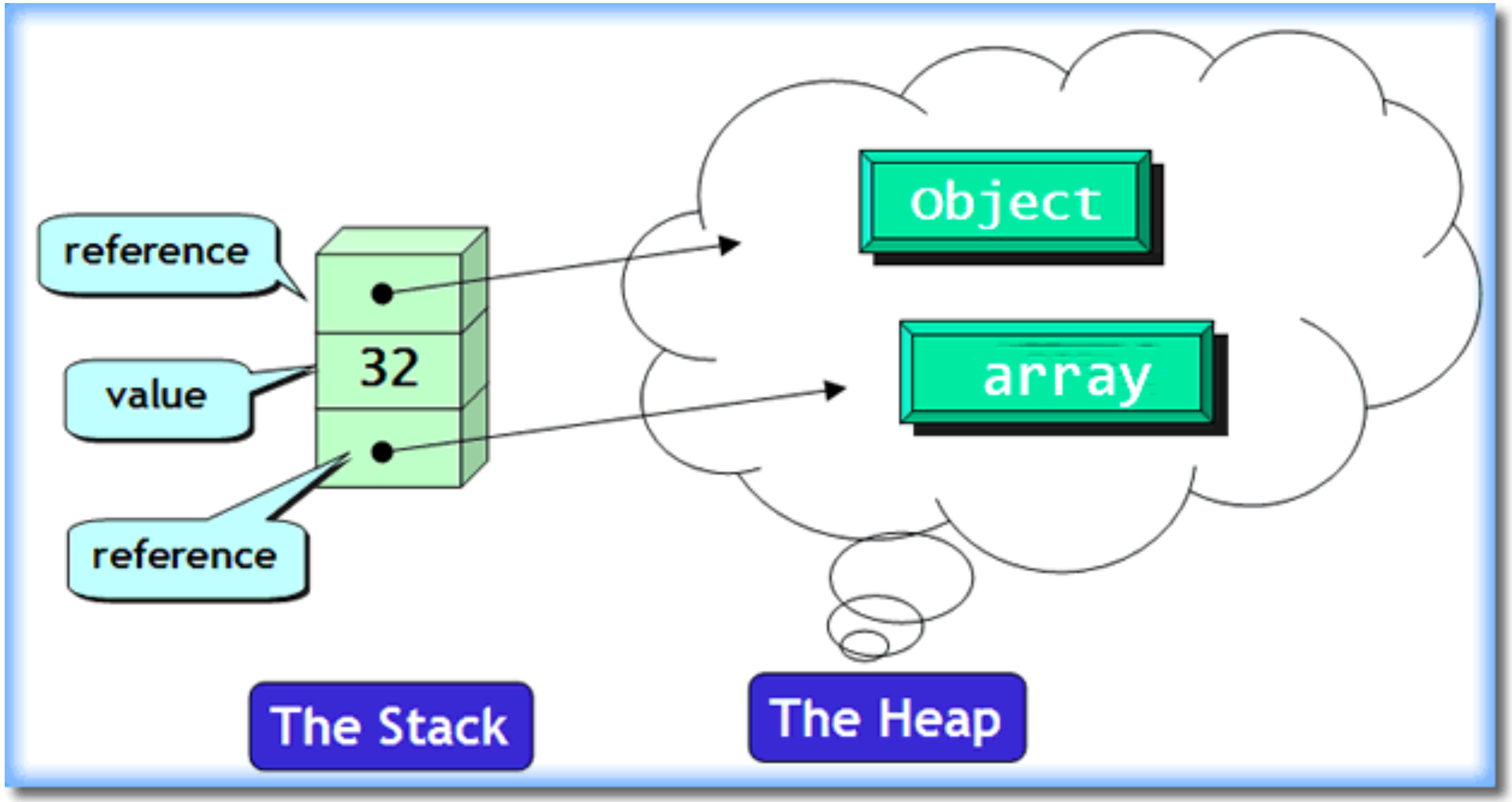


- Data types that work this way are known as reference types
- Variable of those types are reference variables

Why Declare Variables?

- Different primitive values require different amount of memory
- When declaring a variable, we tell the compiler how much memory to allocate
  - `int` (4 bytes)
  - `double` (8 bytes)
  - …
- In Python everything is an object, thus all variables hold references

Memory Model



Value semantic

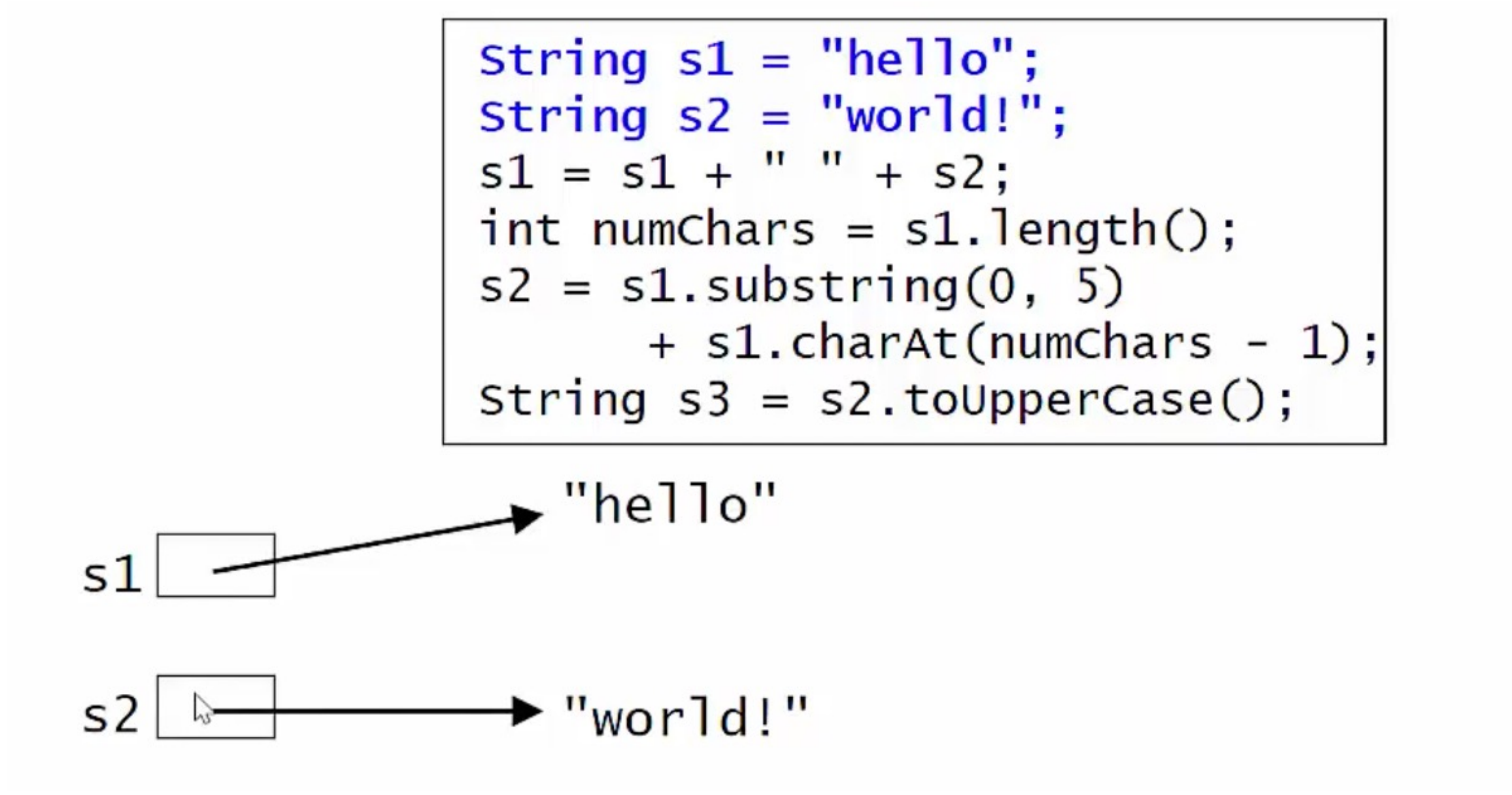
- Value semantics (or value types): behavior where values are copied when assigned, passed as parameters, or returned
- All primitive types in Java use value semantics
- When one variable is assigned to another, its value is copied
- Modifying the value of one variable does not affect others

```
int x = 5;
int y = x;      // x = 5, y = 5
y = 17;         // x = 5, y = 17
x = 8;          // x = 8, y = 17
```

Reference semantic

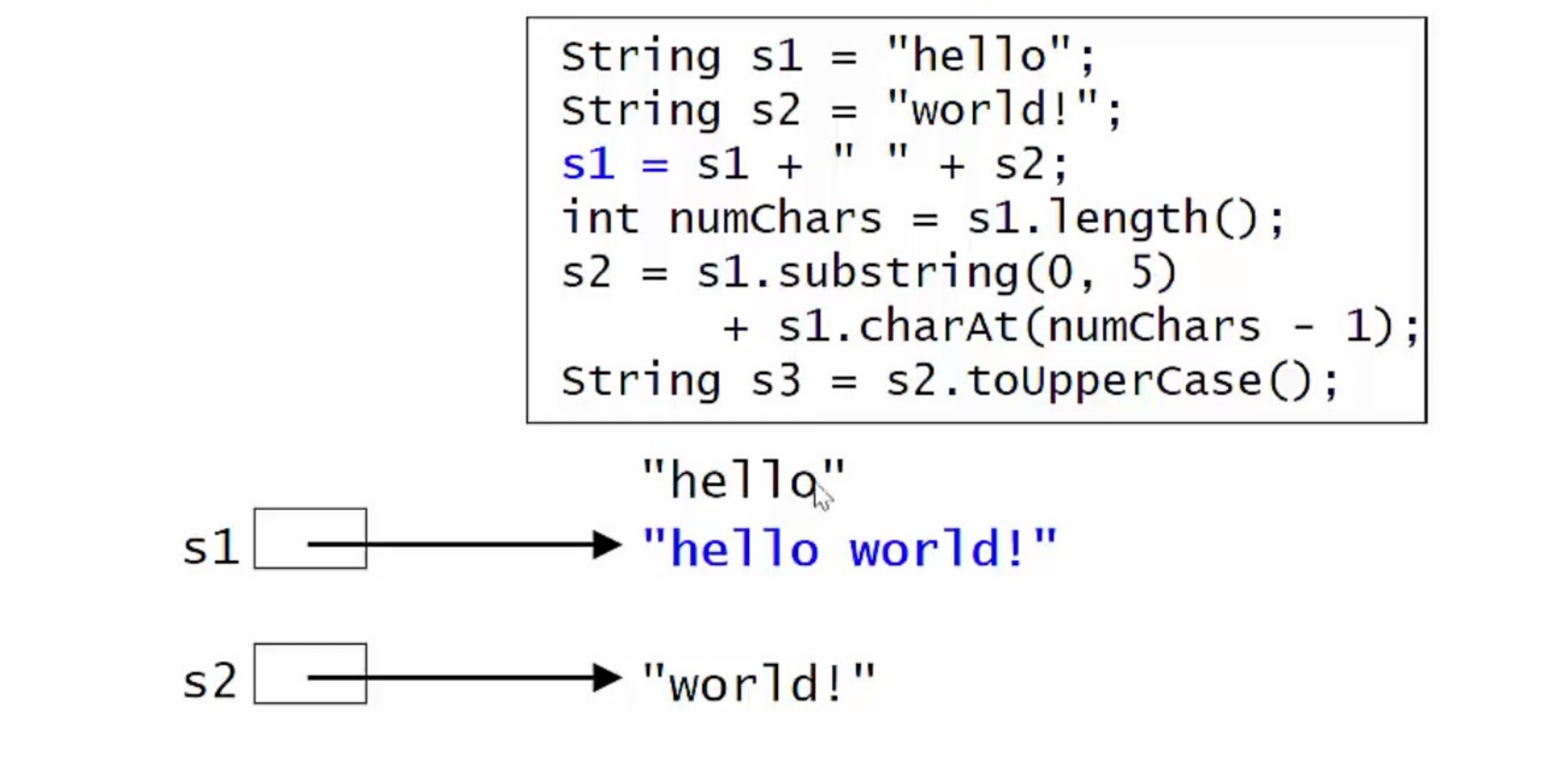
- If a variable represents an object, the object itself is not stored inside the variable
- The object is located somewhere else in memory, and the variable holds the memory address of the object
  - We say that the variable stores a reference to the object
  - Such variables are called reference variables (or types)
- When one variable is assigned to another, the object is not copied; both variables refer to the same object
- Modifying the value of one variable will affect others

The `String` object



```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
       + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

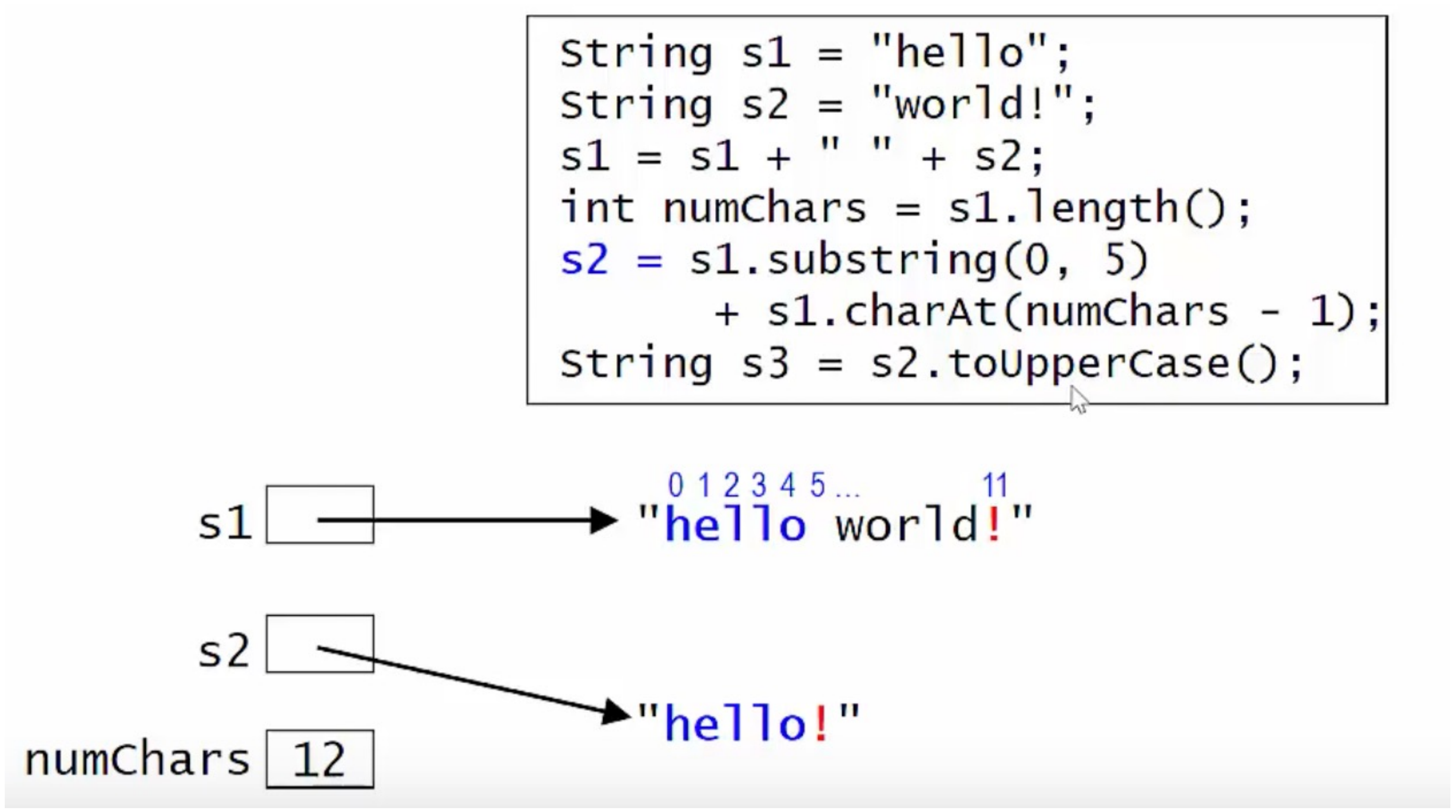
The `String` object



```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- Strings are immutable

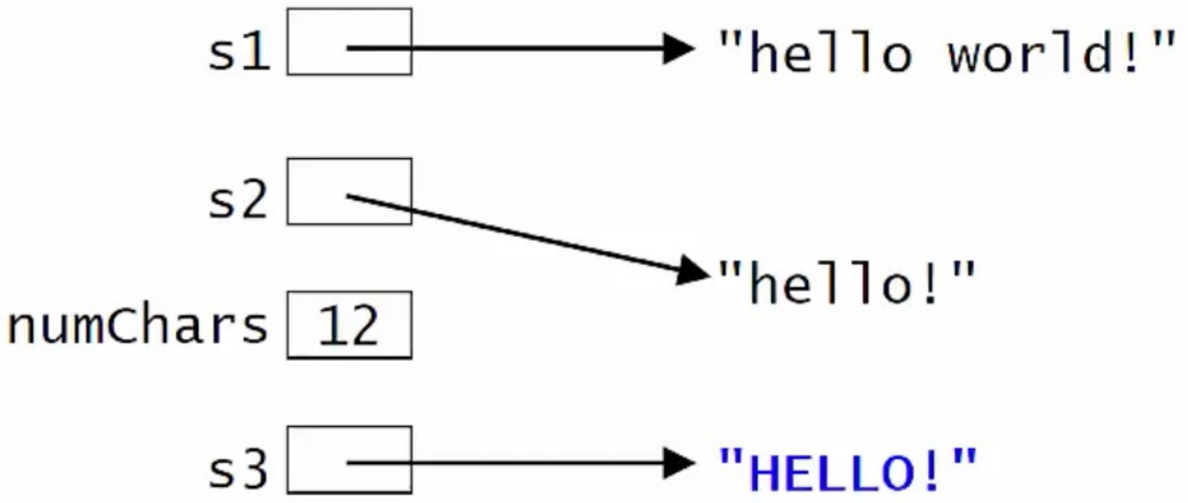
The `String` object



```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
        + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

The `String` object

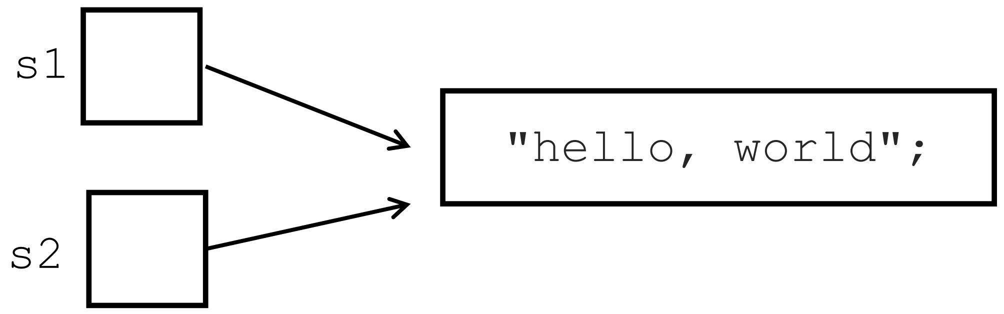
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
        + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



Copying references

- When we assign the value of one reference variable to another, we copy the reference to the object
- We do not copy the object itself

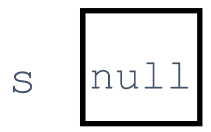
```
String s1 = "hello, world";
String s2 = s1;
```



`null` references

- To indicate that a reference variable doesn't yet refer to any object, we can assign it a special value called `null`

```
String s = null;
```



- Attempting to use a `null` reference to access an object produce a `NullPointerException`
  - Pointer is another name for reference

```
char ch = s.charAt(5); //NullPointerException
```