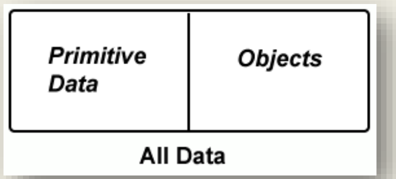
Data

M3 – UF4 – NF2

Data categories



- Small, fixed #bytes
- 8 types
- Cannot be created

- Many bytes
- Data type → Class
- Can be created

Primitive Data Types

byte	short	int	long	float	double	char	boolean

Integer Primitive Data Types

Type	Size	Range
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767
int	32 bits	-2 billion to +2 billion (approximately)
long	64 bits	-9E18 to +9E18 (approximately)

Floating Point Primitive Data Types

Type	Size	Range
float	32 bits	-3.4E38 to +3.4E38
double	64 bits	-1.7E308 to 1.7E308

Curiosity

'S'
'\n'

true
false

"S"

String

Numeric

Variables and Assignments

variable — a named location in main memory which uses a particular data type to hold a value.

Declaration `long payAmount =`

dataType variableName initialValue

Use [A-Za-z0-9$_]
Don't start with digit
No space
Case sensitive
lowerCamelCase

Also...

```
dataType variableName1, variableName2;
dataType vName1 = initialV1,
         vName2 = initialV2;
```

Correct declaration?

```
int myPay, yourPay;
long good-by ;
short shrift = 0;
double bubble = 0, toil= 9, trouble = 8
byte the bullet ;
int  double;
char thisMustBeTooLong ;
int 8ball;
float a=12.3; b=67.5; c= -45.44;
```

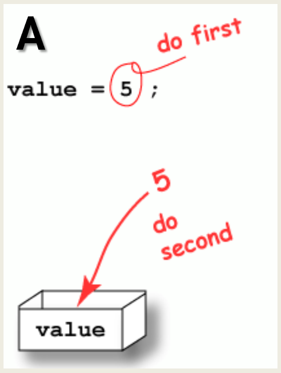
Correct declaration? SOL

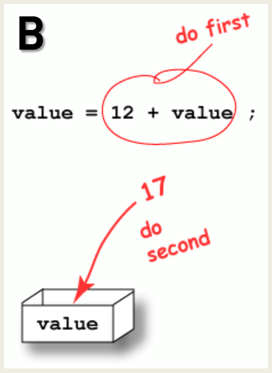
```
int myPay, yourPay; // OK
long good-by ;  // bad identifier: "-" not allowed
short shrift = 0;  // OK
double bubble = 0, toil= 9, trouble = 8 // missing ";" at end.
byte the bullet ;    // bad identifier: can't contain a space
int  double;    // bad identifier: double is a reserved word
char thisMustBeTooLong  ;    // OK in syntax, but a poor choice
                             // for a variable name
int  8ball;    // bad identifier: can't start with a digit
float a=12.3; b=67.5; c= -45.44; // bad syntax: don't use ";" to separate variables
```

Assignment semantics

- It works in 2 steps:
  - Do the calculation / use the value on the RIGHT
  - Replace the content of the variable on the LEFT

```
int value;
value = 5;      //A
value = 12 + value;  //B
```





Expressions and operators

An **expression** is a combination of literals, operators, variable names, and parentheses used to calculate a value.

Arithmetic

Positive/negative numbers

Operator	Meaning	precedence
-	unary minus	highest
+	unary plus	highest
*	multiplication	middle
/	division	middle
%	remainder	middle
+	addition	low
-	subtraction	low

num1 = 5 / 2; **2**

num2 = 1 / 2; **0**

num1 = 5.0 / 2.0; **2.5**

num2 = 1.0 / 2.0; **0.5**

num3 = 1.5 + 7/2; **4.5**

num4 = ((3/2)*5.7) +9/10; **5.7**

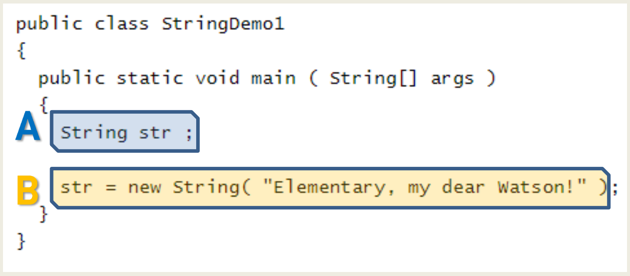
Use balanced **Parentheses** in case of doubt!!!
Even "*nested parentheses*"

Expression Rules

RULE: An integer operation is always done with 32 bits or more. If one or both operand is 64 bits (data type `long`) then the operation is done with 64 bits. Otherwise the operation is done with 32 bits, even if both operands are smaller than 32 bits.

If **both** operands of an arithmetic operator are integers, then the operation is an integer operation. If any operand is floating point, then the operation is floating point.

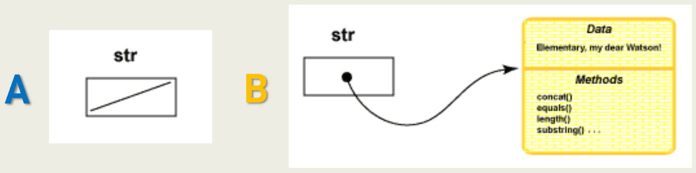
Object Data

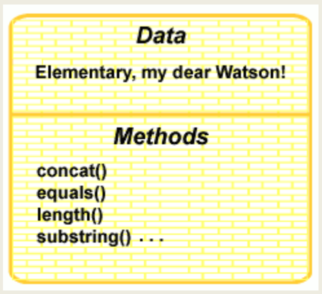


```
public class StringDemo1
{
  public static void main ( String[] args )
  {
A   String str ;

B   str = new String( "Elementary, my dear Watson!" );
  }
}
```

Analogy



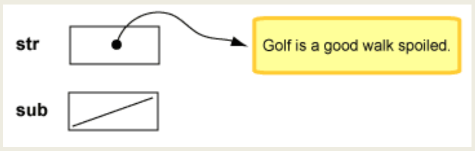


Running an Object Method

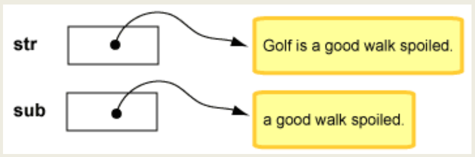
- **Dot Notation**:
  - `objectReference.variableName`
  - `objectReference.methodName(<par>)`

```
len = str.length();    →    len = 27;
```

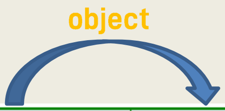
//Mètodes que creen altres objectes
String str = new String("Golf is a good walk spoiled.");



String sub = str.substring(8);



Type Wrappers



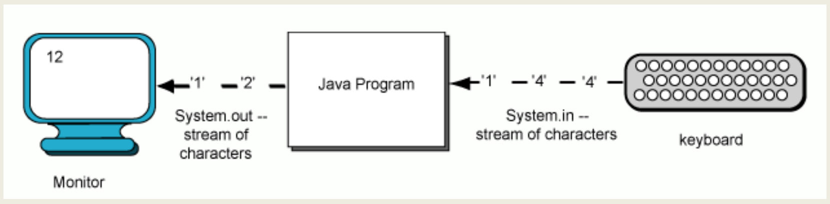
primitive type	Wrapper type
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

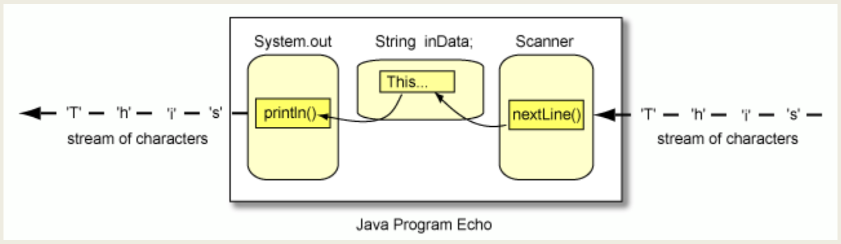
- Automatically imported

```
public class WrapperDemo
{
  public static void main ( String[] args )
  {
    Integer value = new Integer( 103 );        // hold the value 103
                                               // inside an Integer object
    Double dvalue = new Double( -32.78 );      // hold a double precision
                                               // value inside a Double object

    System.out.println( "Integer object holds: " + value );
    System.out.println( "Double  object holds: " + dvalue );
  }
}
```

Input and Output


12
Monitor
'1' '2'
System.out -- stream of characters
Java Program
'1' '4' '4'
System.in -- stream of characters
keyboard



import java.util.Scanner;

System.out.println(text) ⇐ Scanner teclat = new **Scanner**(System.in);
String text = teclat.nextLine();

Numbers as Input

Digits are characters!

In order to collect integers from the input, use:

`nextInt()`

Reads **`String`** of digits → Converts them into **`int`**

– A space or EOL character ends the group

– A non-digit character cannot be part of the group (Throws an Exception and ends the program)

Question

Floating Point

- If you want a floating point input, use:

  `nextFloat()` or **`nextDouble()`**

- **Scientific notation** can be used:

  doubleOrFloatNumber**E**(**+**/**-**)exponential

Examples: `1.314E+1` `-7.001e-2`

*(The **+** simbol is optional)*

Using the Math Class

- We can use directly Class methods (static):

  `Class.method( parameters )`

Example:

```
Math.sqrt(<double>); Math.PI;  Math.cos(<double>);
        Math.pow(<double a>, <double b>);
```

Not Necessary Casting:

```
int x = 9;
System.out.println( Math.sqrt( (double)x );
```

Necessary Casting:

```
int x = 1, y = 9;
System.out.println( Math.sqrt( (double)x/y );
```

First thing done