Literals, Variables, Data types

- Constants in Java are called literals.

- 123
- 123.45

Int
double

- ’a’
- ‘&’

char

- "Gosling“
- “this is a line”

String

- true
- false

boolean

Later on we’ll see another way to define constants…..

Literals, Variables, Data types

Variables

- Named location in memory
- Program controls its contents:
- Initialize, change, display, access
- Java is strongly typed
- You must declare the type of each variable … cannot change this later

Literals, Variables, Data types

Data types: 8 primitive types

Need to know int, double, char, boolean

Numeric

int	100 234 0
long	100L 234L 0L
float	100.12f 234.0f 0.0f
double	100.12 234.0 0.0
char	‘a’, ‘b’, … ‘A’, ’B’,… ‘0’, ‘1’, ‘2’, … ‘~’, ‘@’, ‘#’, …
boolean	true, false

int and *double* are default types for numeric literals

Literals, Variables, Data types

Integer numeric types vary according to

- the amount of memory
- smallest/largest values

data type	memory	minimum value	maximum value
`byte`	1 byte	-128	127
`short`	2 bytes	-32768	32767
`int`	4 bytes	-2147483648	2147483647
`long`	8 bytes	-9223372036854775808	9223372036854775807

Literals, Variables, Data types

Integer numeric types vary according to

Operators for arithmetic expressions

Addition

Subtraction

Multiplication

Division

Modulo

Modulo

operator	example of use	example's result
+	7 + 11	18
-	12 - 5	7
*	3 * 4	12
/	13 / 5	2
%	13 % 5	3

Sample program : IntegerArithmetic.java

Literals, Variables, Data types

Decimal numeric types vary according to

- the amount of memory
- smallest/largest values

data type	memory	maximum
`float`	4 bytes	3.4028235×10^{38}
`double`	8 bytes	$1.7976931348623157 \times 10^{308}$

Literals, Variables, Data types

Decimal - operators for arithmetic expressions

- Addition
- Subtraction
- Multiplication
- Division

operator	example of use	example's result
+	7.1 + 1.1	8.2
-	12.1 - 5.0	7.1
*	2.2 * 2.2	4.84
/	10.0 / 4.0	2.5

Sample program : FuelConsumption.java

Literals, Variables, Data types

Decimal – values are often approximations

Sample program : Approximations.java

boolean

boolean type has 2 values: true, false

Operators && || !

```
boolean xyz = true ;
boolean found ;

If (xyz) System.out.println("the variable is true");

If (xyz && found ) ……..
```

char

char is used for single characters

Usual operators < <= == != > >=

```
char a, b, c;
a = '*';
b = 'q';
c = '1';

If (a == b) ……
```

String

Literals enclosed in double quotes

“this is a line of text”
“any character such as $ % * . < etc can be included”

Variables
Declared as a String type
e.g.

```
String firstName, lastName, address;

firstName = “Jones”;

firstName.length() ;            // see next page
```

String

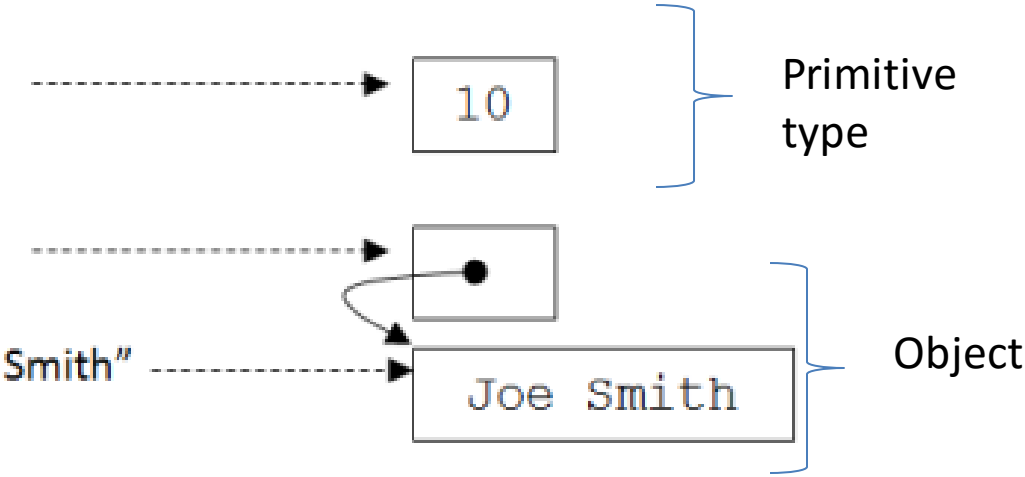
Memory for Strings is handled differently from primitive data types

Java code:

```
int  hoursWorked = 10;
String  fullName = "Joe Smith";
```

The storage location for hoursWorked holds the value 10

The storage location for fullName contains a reference to the storage locations holding the text "Joe Smith"



String

Many useful methods for handling String data

Useful String methods		
method name	type	description
charAt(...)	char	returns the character at a specified position (provided as the argument) where position is one of 0, 1, ..., up to the length of the string.
equals(...)	boolean	used to determine if two strings are identical
equalsIgnoreCase(...)	boolean	used to determine if two strings are identical irrespective of case
indexOf(...)	int	returns the first position of a character provided as an argument, or -1 if it is not present
length()	int	returns the length of a string
toLowerCase()	String	converts all characters to lower case
toUpperCase()	String	converts all characters to upper case
trim()	String	removes leading spaces (blanks) and trailing spaces from a string

Table 2.1: Some of the useful String methods

System.out.**println**(…)

Used to display a line of output on the Terminal Window

What is output for:

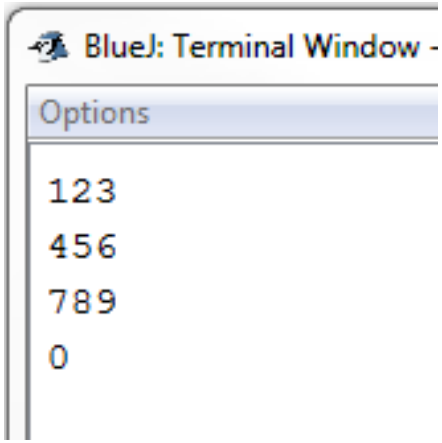
```
public static void main (){
   System.out.println("123");
   System.out.println("456");
   System.out.println("789");
   System.out.println("0");
}
```

System.out.**println**(…)

Used to display a line of output on the Terminal Window

What is output for:

```
public static void main (){
   System.out.println("123");
   System.out.println("456");
   System.out.println("789");
   System.out.println("0");
}
```



System.out.**print**(…)

Used to continue a line displayed on the Terminal Window

What is the output for:

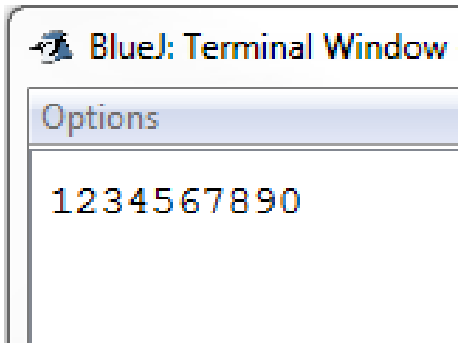
```
public static void main (){
    System.out.print("123");
    System.out.print("456");
    System.out.print("789");
    System.out.print("0");
}
```

System.out.**print**(…)

Used to continue a line displayed on the Terminal Window

What is the output for:

```
public static void main (){
    System.out.print("123");
    System.out.print("456");
    System.out.print("789");
    System.out.print("0");
}
```



System.out….(…)

What is the output for:

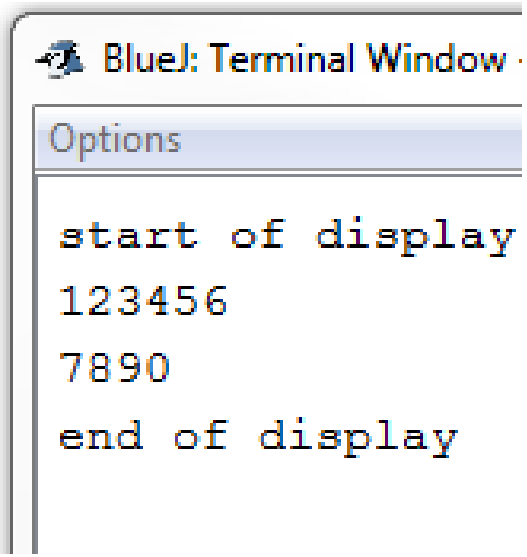
```
public static void main (){
    System.out.println("start of display");
    System.out.print("123");
    System.out.println("456");
    System.out.print("789");
    System.out.println("0");
    System.out.println("end of display");
}
```

System.out....(...)

Sometimes Important for labs and assignments
- But not for a test/exam

What is the output for:

```
public static void main (){
    System.out.println("start of display");
    System.out.print("123");
    System.out.println("456");
    System.out.print("789");
    System.out.println("0");
    System.out.println("end of display");
}
```



Expressions

	Some Java Expressions
1	`32.0`
2	`9.0 / 5.0`
3	`105 % 10`
4	`9.0 / 5.0 * c`
5	`9.0 / 5.0 * c + 32.0`

Operator Priorities Highest to Lowest
`* / %`
`+ -`

If there is more than one operator at the same priority the evaluation goes from left to right.

Suppose c=2.0;
As if:

$$\underbrace{\underbrace{\underbrace{(\,(9.0 / 5.0)}_{1.8} * c\,)}_{3.6} + 32.0}_{35.6}$$

Priorities (and so order of evaluation) can be overridden with a subexpression … operations enclosed in parentheses

A sub-expression is always evaluated before the expression in which it is contained is evaluated. Of course the sub-expression is evaluated according to the rules of expressions.

Expressions

Suppose c=2.0;

Consider the following, coded with sub-expressions:

(9.0 / (5.0 * (c + 32.0)))

34.0

170.0

0.0529

Mixed Mode Expressions

A **mixed-mode** expression is an expression that involves `int`s and `double`s

Example: 9 / 5.0 * 2 + 32

If an operation involves two operands where one is an `int` and the other is a `double`, then the `int` is **converted automatically** to its `double` equivalent **before** the operation occurs.

9 / 5.0 * 2 + 32

1. The first operation is "/" … the 9 is converted to 9.0 and we have
   9.0/5.0 * 2 + 32
   And so we have
   1.8 * 2 + 32
2. "*" is performed next … the 2 is converted to 2.0
   1.8 * 2.0 + 32
   And so we have
   3.6 + 32
3. 32 becomes 32.0 and we have the final result of 35.6

Mixed Mode Expressions

Example: 9 / 5 * 2 + 32.0

9 / 5 * 2 + 32.0

1. The first operation is "/" … 9/5 is 1
   And so we have
   1 * 2 + 32.0
2. "*" is performed next
   And so we have
   2 + 32.0
3. 2 becomes 2.0 and we have the final result of 34.0

This loss of precision can have drastic affect on results