



جامعة بيروت العربية
BEIRUT ARAB UNIVERSITY

CMPS 241

Introduction to Programming

println Statement, Primitive Data Types, Expressions

Data types

- **Type:** A **name for a category** or **set of data values** that are related, as in type `int` in java, which used to represent **integer** values.
 - Constrains the **operations that can be performed on data**
 - Many languages ask the programmer to specify types
 - Examples: integer, real number, string
- Internally, computers store everything as 1s and 0s
 - `104` → 01101000
 - `h` → 01101000
 - `i` → 01101001
 - `"hi"` → 0110100001101001
- Character representation
 - ASCII Code → 7 bits
 - Extended ASCII Code → 8 bits
 - Unicode → 16 bits

ASCII

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(72	H	104	h
41)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	
61	=	93]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	Ł	224	Ó
129	ü	161	í	193	ł	225	ô
130	é	162	ó	194	Ł	226	Ô
131	â	163	ú	195	ł	227	Ò
132	ä	164	ñ	196	—	228	õ
133	à	165	Ñ	197	†	229	Õ
134	ä	166	ª	198	ä	230	µ
135	ç	167	º	199	Ä	231	þ
136	ê	168	¿	200	Ł	232	þ
137	ë	169	®	201	ƒ	233	Ú
138	è	170	™	202	Ł	234	Û
139	ï	171	½	203	ƒ	235	Ü
140	î	172	¼	204	ƒ	236	ý
141	ì	173	¡	205	=	237	Ý
142	Ä	174	«	206	ƒ	238	—
143	Å	175	»	207	¤	239	'
144	É	176	☐	208	ð	240	≡
145	æ	177	☐	209	Đ	241	±
146	Æ	178	☐	210	Ê	242	≡
147	ø	179		211	Ë	243	¾
148	ö	180	†	212	È	244	¶
149	ò	181	Á	213	Ì	245	§
150	ú	182	Â	214	Í	246	÷
151	ù	183	Ã	215	Î	247	°
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	ƒ	217	ƒ	249	°
154	Ü	186		218	ƒ	250	°
155	ø	187	ƒ	219	■	251	°
156	£	188	ƒ	220	■	252	°
157	Ø	189	¢	221	¡	253	°
158	×	190	¥	222	¡	254	■
159	f	191	ƒ	223	■	255	nbsp

Java's primitive types

- **primitive types**: there are 8 simple types for numbers, text, etc.
 - Java also has **object types**, which we'll talk about later
- The most commonly used types

Type	Description	Examples
<code>int</code>	<code>integers</code> (up to $2^{31} - 1$)	42, -3, 0, 926394
<code>double</code>	<code>real numbers</code> (up to 10^{308})	3.1, -0.25, 9.4e3
<code>char</code>	<code>single text characters</code>	'a', 'X', '?', '\n'
<code>boolean</code>	<code>logical values</code>	true, false

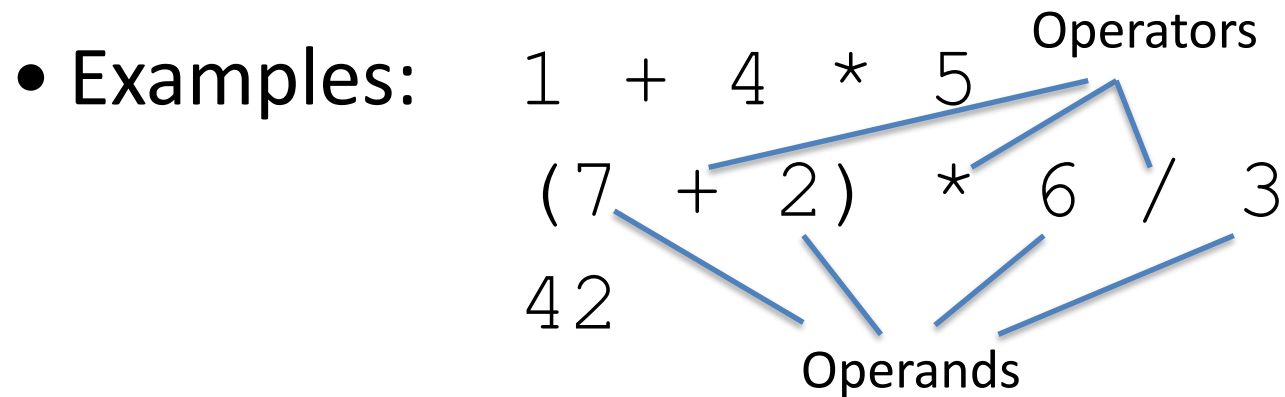
- Why does Java distinguish integers vs. real numbers?

Java's primitive types

Type	Description	Size
<code>int</code>	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
<code>byte</code>	The type describing a single byte , with range -128 . . . 127	1 byte
<code>short</code>	The short integer type, with range -32768 . . . 32767	2 bytes
<code>long</code>	The long integer type , with range -9,223,372,036,854,775,808 . . . -9,223,372,036,854,775,807	8 bytes
<code>double</code>	The double-precision floating-point type , with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
<code>float</code>	The single-precision floating-point type , with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
<code>char</code>	The character type, representing code units in the Unicode encoding scheme	2 bytes
<code>boolean</code>	The type with the two truth values <code>false</code> and <code>true</code>	1 bit

Expressions

- **Expression:** A **value** or **operation** that computes a value.



- The simplest expression is a ***literal value*** such as 42 or 28.9.
- A complex expression can use **operators**, **operands** and **parentheses**.

Arithmetic operators

- **Operator:** Combines multiple **operands** (values) or expressions.
 - + addition
 - subtraction (or negation)
 - * multiplication
 - / division
 - % modulus (a.k.a. remainder)
- **Evaluation:** The process of obtaining the value of an expression
 - As a program runs, its expressions are *evaluated*.
 - `1 + 1` evaluates to 2
 - `System.out.println(3 * 4);` prints 12
 - How would we print the text `3 * 4` ?

Integer division with /

- When we divide integers, the **quotient** is also an integer.

- $14 / 4$ is 3, not 3.5

$$\begin{array}{r} 3 \\ 4 \overline{)14} \\ \underline{12} \\ 2 \end{array}$$

$$\begin{array}{r} 4 \\ 10 \overline{)45} \\ \underline{40} \\ 5 \end{array}$$

$$\begin{array}{r} 52 \\ 27 \overline{)1425} \\ \underline{135} \\ 75 \\ \underline{54} \\ 21 \end{array}$$

- More examples:

- $32 / 5$ is 6
- $84 / 10$ is 8
- $156 / 100$ is 1

– Dividing by 0 causes a **run-time error** when your program runs.

Integer remainder with %

- The % operator computes the **remainder** from integer division.

- $14 \% 4$ is 2

- $218 \% 5$ is 3

$$\begin{array}{r} 3 \\ 4 \overline{) 14} \\ \underline{12} \\ 2 \end{array}$$

$$\begin{array}{r} 43 \\ 5 \overline{) 218} \\ \underline{20} \\ 18 \\ \underline{15} \\ 3 \end{array}$$

What is the result?

$$45 \% 6$$

$$2 \% 2$$

$$8 \% 20$$

$$11 \% 0$$

- Applications of % operator:

- Obtain **last digit** of a number: $230857 \% 10$ is **7**

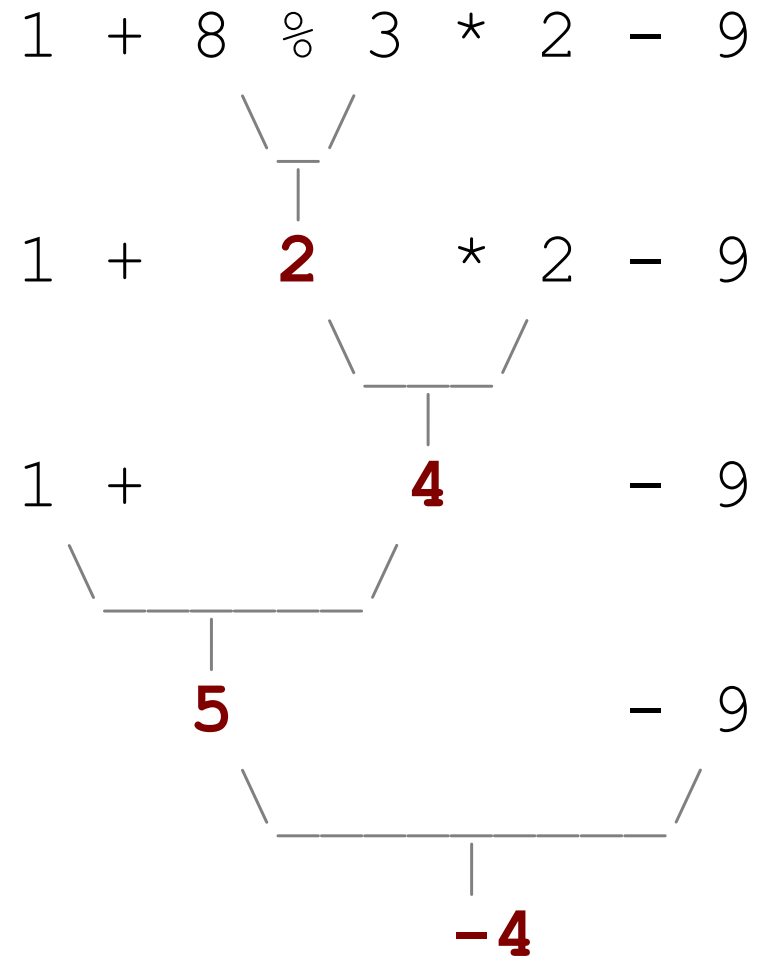
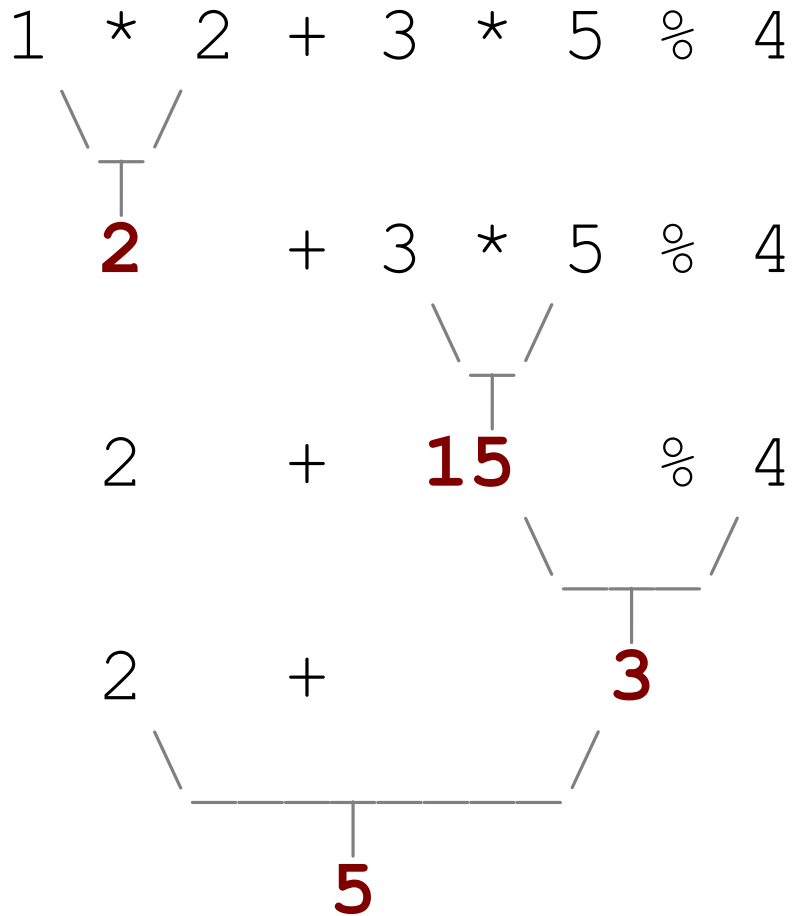
- Obtain **last 4 digits**: $658236489 \% 10000$ is **6489**

- See **whether a number is odd or even**: $7 \% 2$ is **1**, $42 \% 2$ is **0**

Precedence

- **Precedence:** Order in which operators are evaluated.
 - Generally operators evaluate *left-to-right*.
 $1 - 2 - 3$ is $(1 - 2) - 3$ which is -4
 - But $*$ $/$ $\%$ have a higher *priority* (level of precedence) than $+$ $-$
 $1 + 3 * 4$ is 13
 $6 + 8 / 2 * 3$
 $6 + 4 * 3$
 $6 + 12$ is 18
 - **Parentheses** can force a certain order of evaluation:
 $(1 + 3) * 4$ is 16
 - Spacing does not affect order of evaluation
 $1 + 3 * 4 - 2$ is 11

Precedence examples



Precedence questions

- What values result from the following expressions?
 - $9 / 5$
 - $695 \% 20$
 - $7 + 6 * 5$
 - $7 * 6 + 5$
 - $248 \% 100 / 5$
 - $6 * 3 - 9 / 4$
 - $(5 - 7) * 4$
 - $6 + (18 \% (17 - 12))$

Precedence questions

- What values result from the following expressions?
 - $9 / 5 = 1$
 - $695 \% 20 = 15$
 - $7 + 6 * 5 = 37$
 - $7 * 6 + 5 = 47$
 - $248 \% 100 / 5 = 9$
 - $6 * 3 - 9 / 4 = 16$
 - $(5 - 7) * 4 = -8$
 - $6 + (18 \% (17 - 12)) = 9$

Real numbers (type double or float)

- Examples: 6.022, -42.0, 2.143e17
 - Placing .0 or . after an integer makes it a **double**.
(e.g. 3 is *int* but 3.0 or 3. *is double*)
- The operators + - * / % () all still work with **double**.
 - / produces an exact answer: 15.0 / 2.0 is 7.5
 - Precedence is the same: () before * / % before + -

Real number example

$$2.0 * 2.4 + 2.25 * 4.0 / 2.0$$



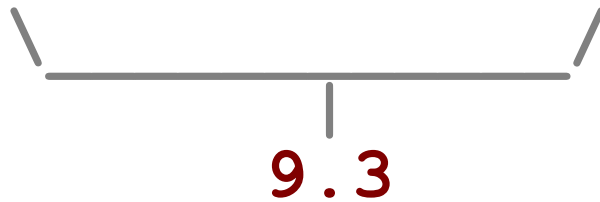
$$4.8 + 2.25 * 4.0 / 2.0$$



$$4.8 + 9.0 / 2.0$$



$$4.8 + 4.5$$

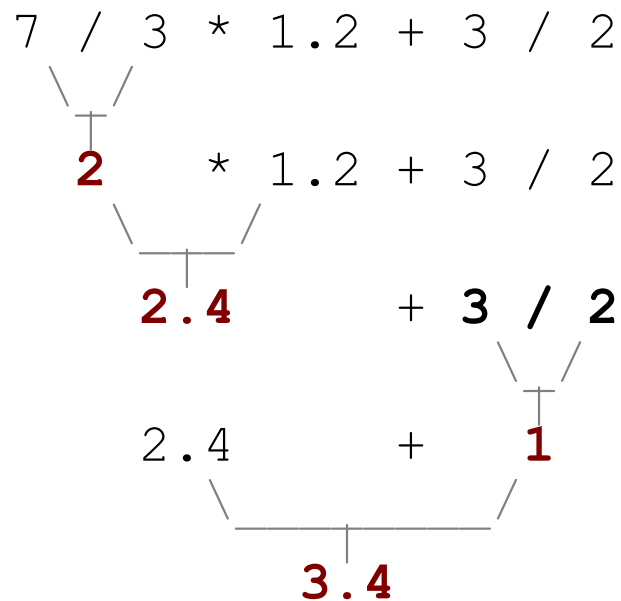


Mixing types

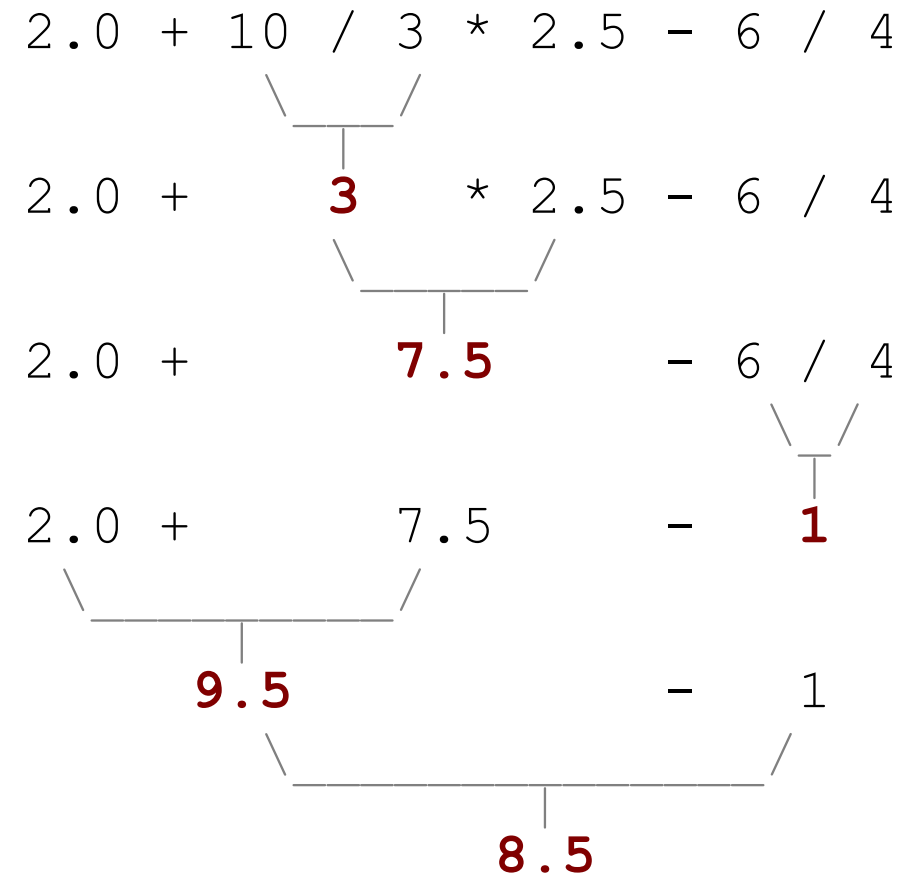
- When **int** and **double** are **mixed**, the result is a **double**.

– $4.2 * 3$ is 12.6

- The conversion is per-operator, affecting only its operands.



– $3 / 2$ is 1 above, not 1.5 .



String concatenation

- **string concatenation:** Using **+** between a string and another value to make a longer string.

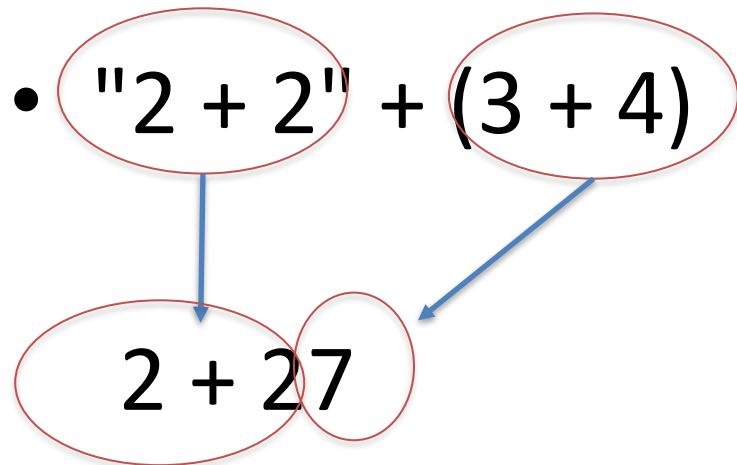
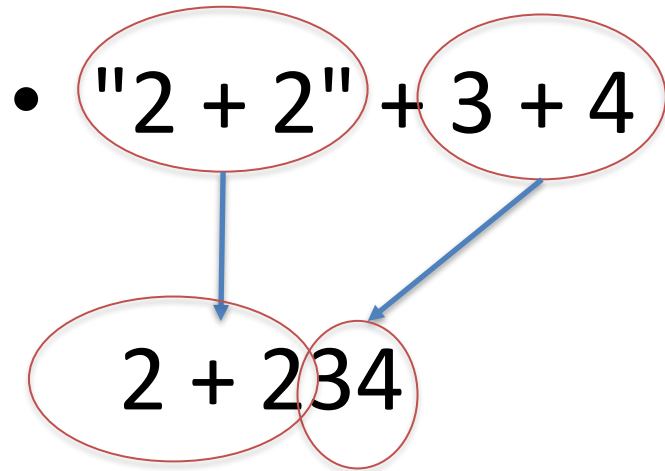
"hello" + 42	is	"hello42"
1 + "abc" + 2	is	"1abc2"
"abc" + 1 + 2	is	"abc12"
1 + 2 + "abc"	is	"3abc"
"abc" + 9 * 3	is	"abc27"
"1" + 1	is	"11"
4 - 1 + "abc"	is	"3abc"

- Use **+** to print a string and an expression's value together.
 - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
 - **Output:** Grade: 83.5

More Examples about Concatenation

- Trace the evaluation of the following expressions, and give their resulting values:
- `"2 + 2" + 3 + 4`
- `"2 + 2 " + (3 + 4)`
- `2 + "(int) 2.0" + 2 * 2 + 2`

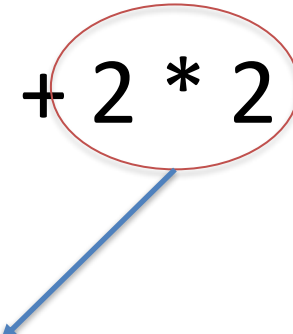
More Examples about Concatenation



More Examples about Concatenation

- $2 + \text{"(int) 2.0"} + 2 * 2 + 2$

2(int) 2.042



Variables

Receipt example

What's **bad** about the following code?

```
public class Receipt {  
    public static void main(String[] args) {  
        // Calculate total owed, assuming 8% tax / 15% tip  
        System.out.println("Subtotal: ");  
        System.out.println(38 + 40 + 30);  
        System.out.println("Tax: ");  
        System.out.println((38 + 40 + 30) * .08);  
        System.out.println("Tip: ");  
        System.out.println((38 + 40 + 30) * .15);  
        System.out.println("Total: ");  
        System.out.println(38 + 40 + 30 +  
                            (38 + 40 + 30) * .08 +  
                            (38 + 40 + 30) * .15);  
    }  
}
```

- The subtotal expression $(38 + 40 + 30)$ is **repeated**
- So **many** `println` statements

Variables

- **Variable:** A piece of the computer's memory that is given a **name** and **type**, and **can store a value**.
 - Like preset stations on a car stereo, or cell phone speed dial:



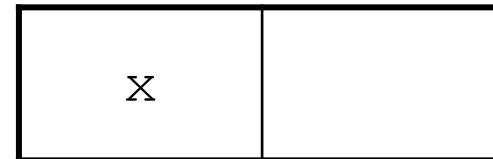
- The **type** tells us **what we can do with the variables**
 - For example, we can compute the sum of two integers
- Steps for using a variable:
 - **Declare** it - state its **name** and **type**
 - **Initialize** it - store a value into it (**assign a value to it**)
 - **Use** it - print it or use it as part of an expression

Declaration

- **variable declaration**: Sets aside memory for storing a value.
 - Variables **must be declared** before they can be used.

***Remember:** must start with a letter or _ or \$ and subsequent characters can be any of those or a number*

- Syntax: **type name;**
 - The name is an *identifier*.



```
int x;
```



```
double myGPA;
```


Assignment

- **assignment:** Stores a value into a variable.
 - The **=** operator is called **assignment operator**
 - On the left you need **variable** name;
 - The **right-hand side** can be **value** or **expression**.

- Syntax: **name = expression;**

```
int x;
```

```
x = 3;
```

x	3
---	---

```
double myGPA;
```

```
myGPA = 1.0 + 2.25;
```

myGPA	3.25
-------	------

Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

type name = value;

```
double myGPA = 3.95;
```

myGPA	3.95
-------	------

```
int x = (11 % 3) + 12;
```

x	14
---	----

Using variables

- Once given a value, a variable can be used in expressions:

```
int x;  
x = 3;  
System.out.println("x is " + x);  
System.out.println(5 * x - 1);
```

// x is 3
// 5 * 3 - 1
// 14

- You can assign a value more than once:

```
int x;  
x = 3;  
System.out.println(x + " here");
```

// 3 here

```
x = 4 + 7;  
System.out.println("now x is " + x);
```

// now x is 11

x	11
---	----

Assignment and algebra

- Assignment uses **=**, but it is not an algebraic equation.
= means, "*store the value at right in variable at left*"
- The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

x	5
---	---

```
int x = 3;
```

```
x = x + 2;    // ???
```

Assignment and types

- A variable **can only store a value of its own type**.

```
int x = 2.5;    // ERROR: incompatible types
```

- An **int** value **can be stored in a double** variable.
 - The **value is converted** into the equivalent real number.

```
double myGPA = 4;
```

myGPA	4.0
-------	-----

```
double avg = 11 / 2;
```

avg	5.0
-----	-----

- Why does avg store 5.0 and not 5.5 ?

Compiler errors

- A variable **can't be used** until it is assigned a value.

```
- int x;  
  System.out.println(x);    // ERROR: x has no value
```

- You may not declare the same variable twice.

```
int x;  
int x;                        // ERROR: x already exists
```

```
int x = 3;  
int x = 5;                    // ERROR: x already exists
```

- How can this code be fixed?

Printing a variable's value

- Use **+** to **print a string and a variable's value on one line.**

```
double grade = (95.1 + 71.9 + 82.6) / 3.0;  
System.out.println("Your grade was " + grade);
```

```
int students = 11 + 17 + 4 + 19 + 14;  
System.out.println("There are " + students +  
                    " students in the course.");
```

- Output:

```
Your grade was 83.2
```

```
There are 65 students in the course.
```