FCDS

Programming I

Lecture 2: Primitive Data Types, Expressions and Variables

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
```

- ASCII Code → 7 bits
- Extended ASCII Code → 8 bits
- Unicode → 16 bits

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	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	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
`int`	integers	(up to 2^{31} - 1)	`42, -3, 0, 926394`
`double`	real numbers	(up to 10^{308})	`3.1, -0.25, 9.4e3`
`char`	single text characters		`'a', 'X', '?', '\n'`
`boolean`	logical values		`true, false`

- Why does Java distinguish integers vs. real numbers?

Java's primitive types

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

Expressions

- **Expression**: A value or operation that computes a value.
  - Examples:
    ```
    1 + 4 * 5
    (7 + 2) * 6 / 3
    42
    ```
  - 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`

```
        3               4                  52
4 ) 14         10 ) 45         27 ) 1425
    12              40               135
     2               5                 75
                                       54
                                       21
```

- 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`

```
        3
4 ) 14
    12
     2
```

```
         43
5 ) 218
    20
     18
     15
      3
```

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

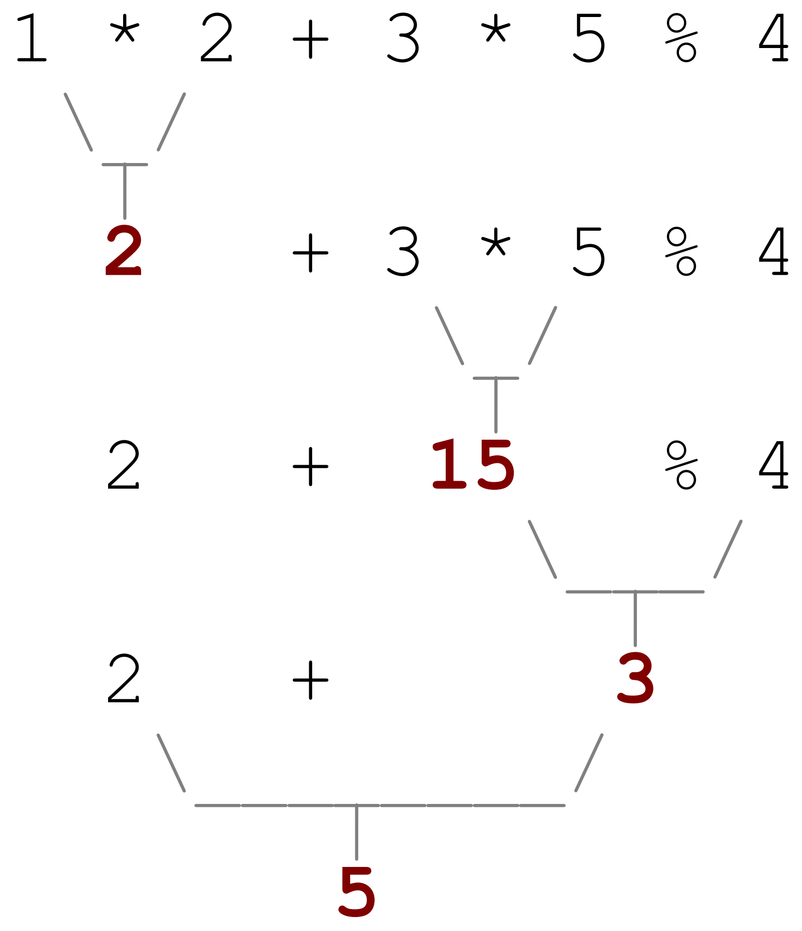
- **Precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`
  - But `* / %` have a higher level of precedence than `+ -`
  - When two operators share an operand the operator with the higher *precedence* goes first
    - `1 + 3 * 4` is `13`
  - When two operators with the same precedence the expression is evaluated *left to right.*
    - `6 + 8 / 2 * 3`
    - `6 + 4 * 3`
    - `6 + 12` is `18`

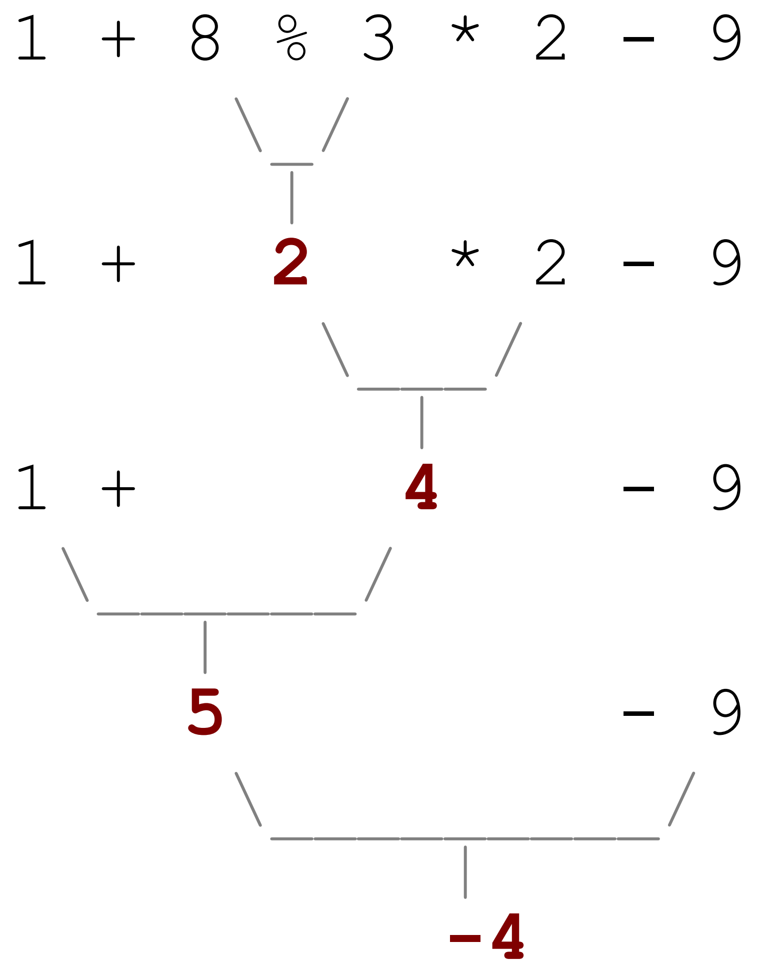
- `6 + 8 * 2 / 3`
- `6 + 16 / 3`
- `6 + 5` is `11`

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 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

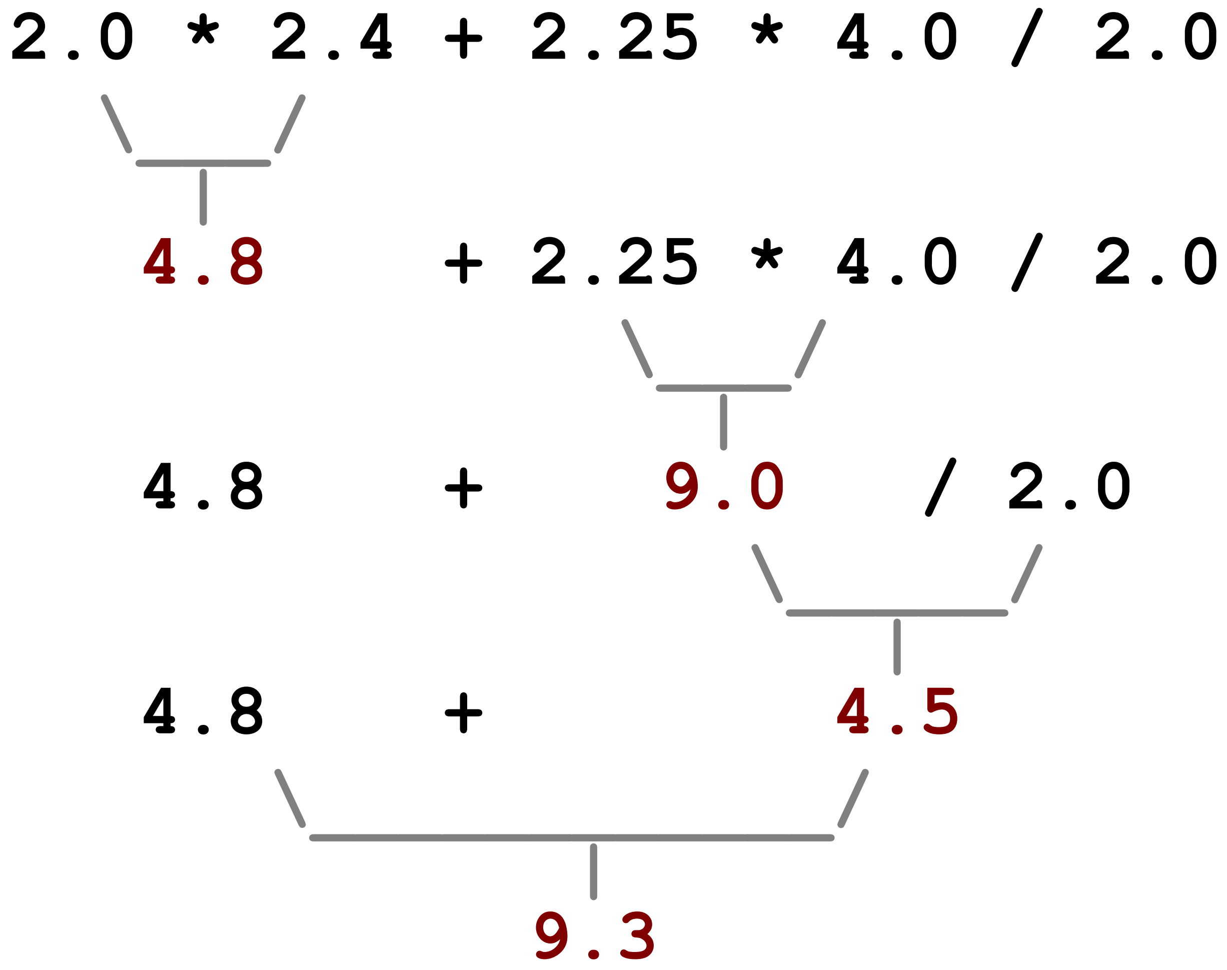
- What values result from the following expressions?
  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`

Real numbers (type `double` or `float`)

- Examples: **`6.022, -42.0, 2.143e17`**
  - Placing **`.0`** or **`.`** after an integer makes it a **`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

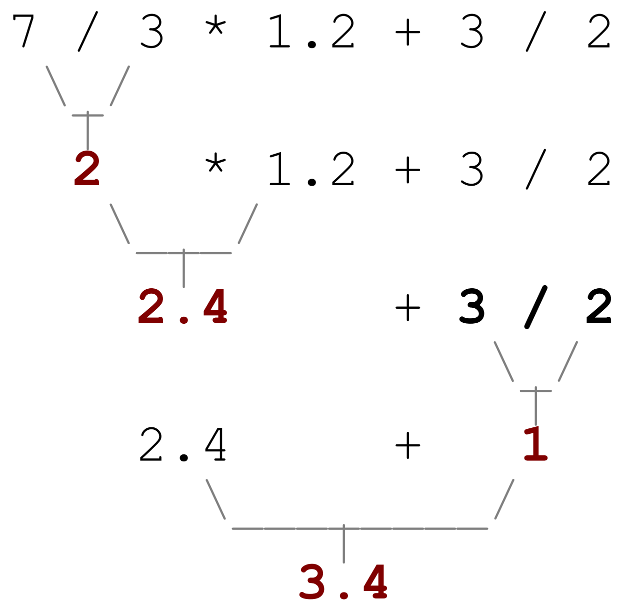


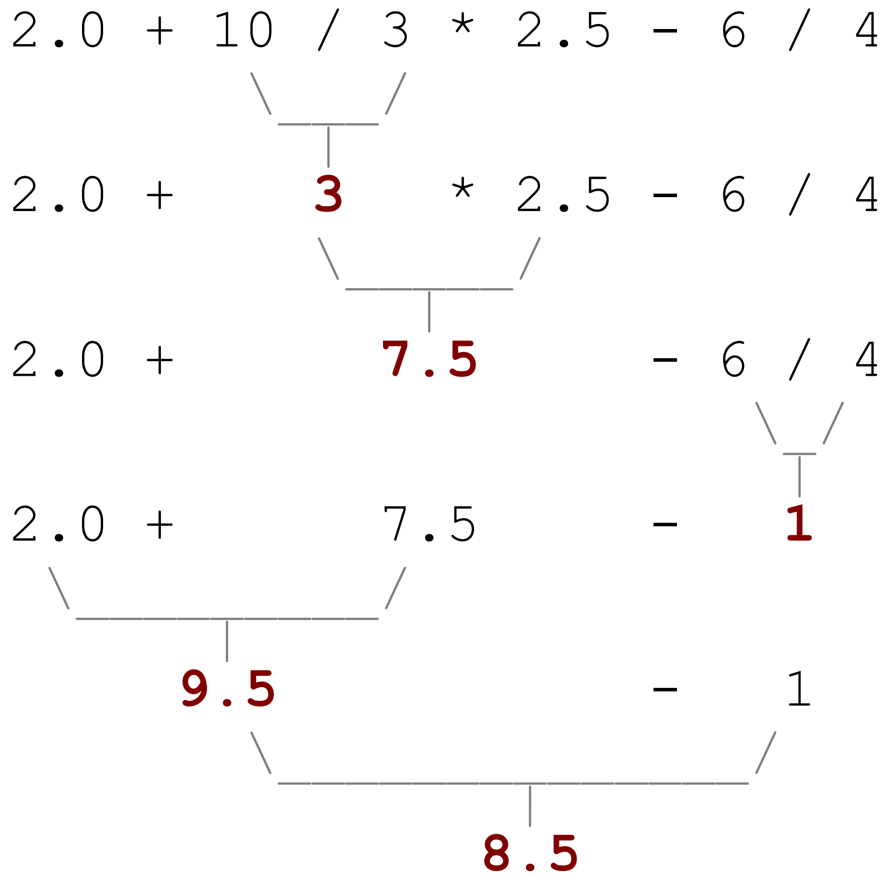
```
2.0 * 2.4 + 2.25 * 4.0 / 2.0

   4.8    + 2.25 * 4.0 / 2.0

   4.8    +     9.0    / 2.0

   4.8    +         4.5

             9.3
```

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`

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:

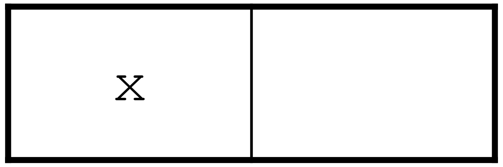
  - The **type** tells us what we can do with the variables
    - For example, we can compute the sum of two integers
  - Steps for using a variable:
    - ***Declare*** it - state its name and type
    - ***Initialize*** it - store a value into it (assign a value to it)
    - ***Use*** it - print it or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

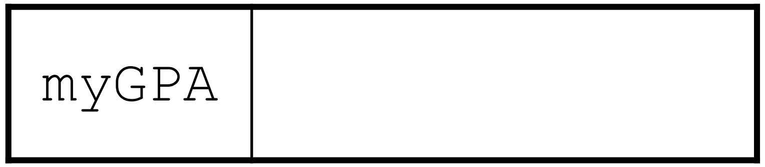
Declaration

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

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

```
int x;
```





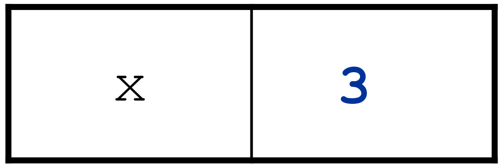
```
double myGPA;
```

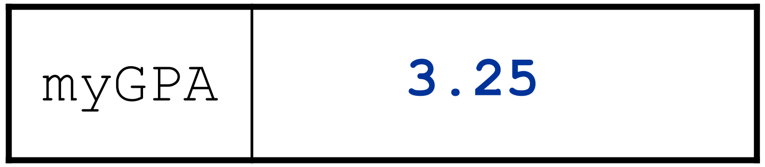
Assignment

- **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;

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





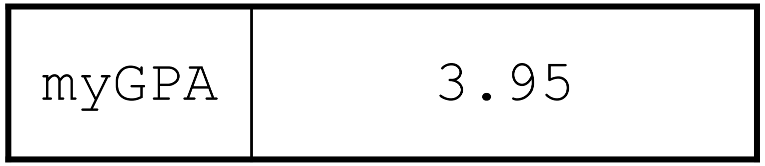
Declaration/initialization

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

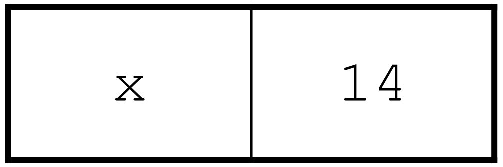
- Syntax:

  type name = **value**;

  - `double myGPA = 3.95;`



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

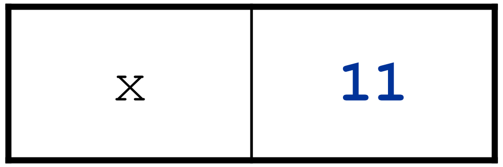


Using variables

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

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

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?

```
int x = 3;
x = x + 2;    // ???
```

x	5

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.
```

Increment and decrement

shortcuts to increase or decrease a variable's value by 1 using unary operators (++ and --)

Shorthand	Equivalent longer version
variable`++;`	**variable** `=` **variable** `+ 1;`
variable`--;`	**variable** `=` **variable** `- 1;`

```
int x = 2;
x++;                // x = x + 1;
                    // x now stores 3

double gpa = 2.5;
gpa--;              // gpa = gpa - 1;
                    // gpa now stores 1.5
```

Modify-and-assign

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += value;	**variable = variable + value;**
variable -= value;	**variable = variable - value;**
variable *= value;	**variable = variable * value;**
variable /= value;	**variable = variable / value;**
variable %= value;	**variable = variable % value;**

```
x += 3;               // x = x + 3;
gpa -= 0.5;           // gpa = gpa - 0.5;
number *= 2;          // number = number * 2;
```

Java Operator Precedence

Description	Operators
Unary Operators	++, --, +, - **Highest**
Binary Multiplicative Operators	*, /, %
Binary Additive Operators	+, -
Assignment Operators	=, +=, -=, *=, /=, %= **Lowest**

- Binary Operators in the same level (such as + and -) are of equal priority and are evaluated left to right. (Example: x * y / 3)
- Unary Operators in the same level (such as + and -) are of equal priority and are evaluated right to left. (Example: ++x - ++y)
- Assignment Operators in the same level (such as =) are of equal priority and are evaluated right to left. (Example: x=y=z=9;)

Example: Evaluate the expression

```
      z – (a + b / 2) + w * -y
      Given  z = 8, a = 3, b = 9, w = 2, y =
-5
            8 – (3 + 9 / 2) + 2 * - -5
         (Step-1)  9/2 = 4
             8 – (3 + 4) + 2 * - -5
         (Step-2)  (3+4) = 7
             8 – 7 + 2 * - -5
         (Step-3)  - - 5 = 5
             8 – 7 + 2 * 5
         (Step-4)  2 * 5 = 10
             8 – 7 + 10
         (Step-5)  8 – 7 = 1
             1 + 10
         (Step-6)  1 + 10 = 11
             11
```

Receipt question

Improve the receipt program using variables.

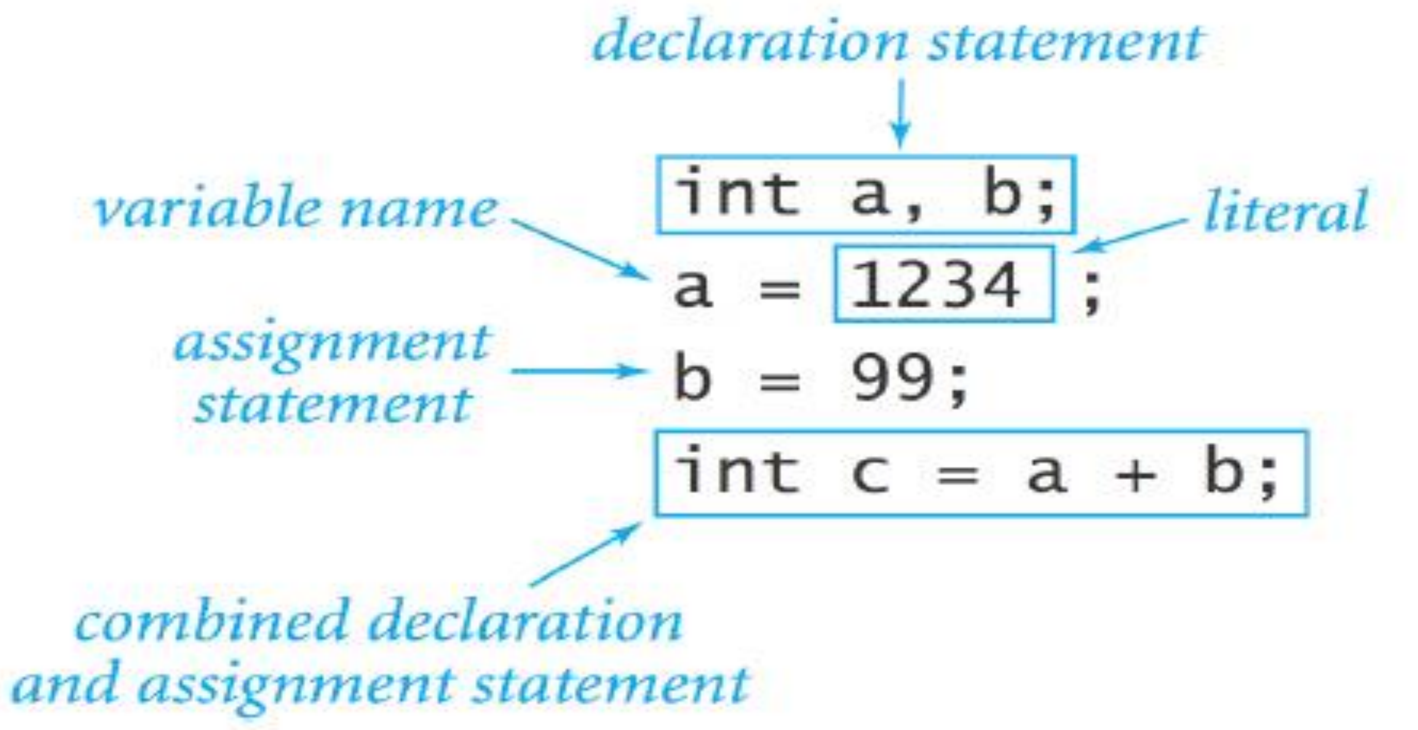
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
         System.out.println("Subtotal:");
         System.out.println(38 + 40 + 30);
         System.out.println("Tax:");
         System.out.println((38 + 40 + 30) * .08);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

Receipt answer

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```

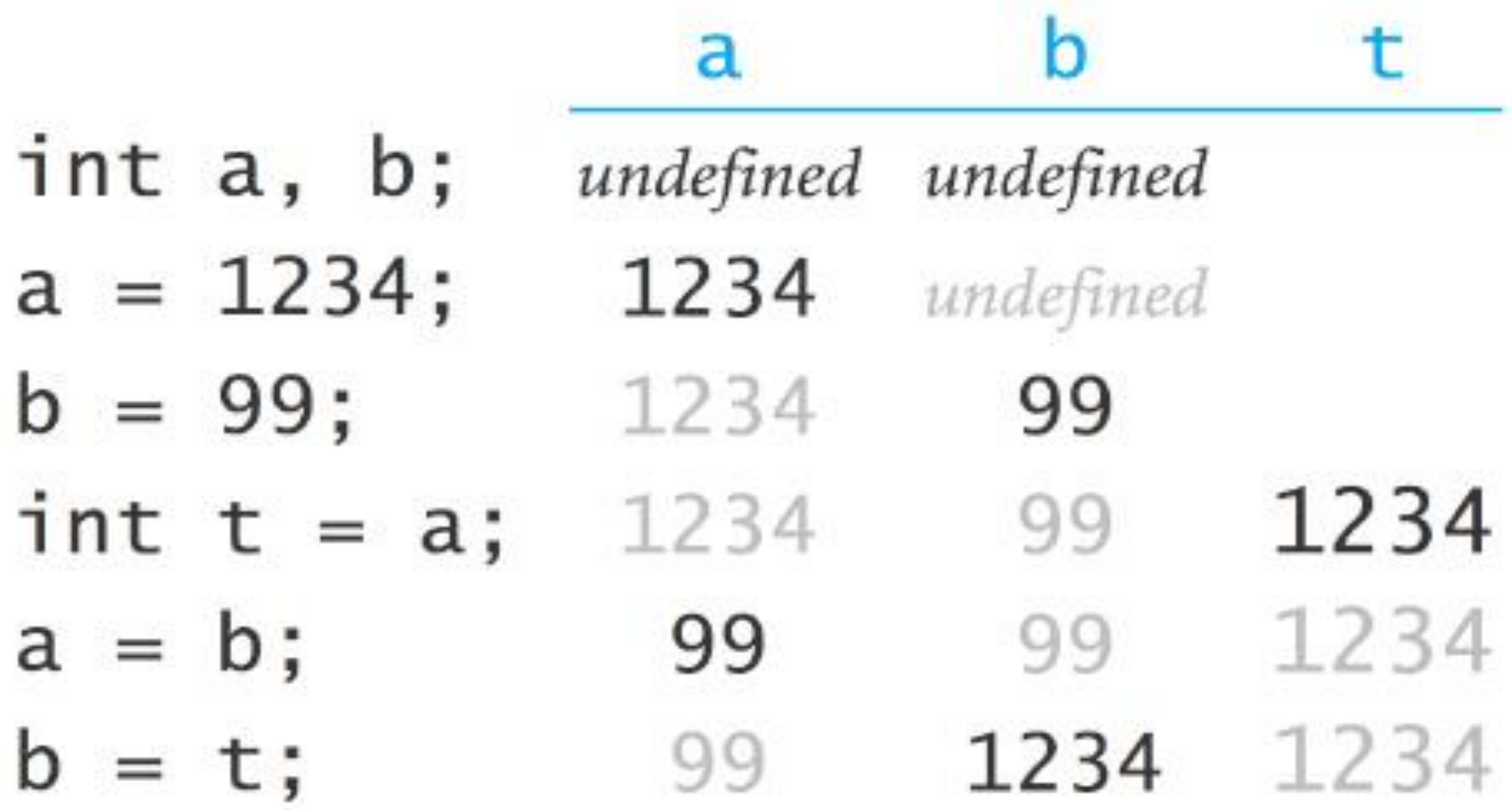
Variables (Summary)

- name, type, value
- declaration and assignment



```
int a, b;
a = 1234 ;
b = 99;
int c = a + b;
```

Trace



	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Type casting

- **Type Cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

- Syntax:

  (**type**) **expression**

  Examples:

```
double result = (double) 19 / 5;        // 3.8
int result2 = (int) result;             // 3
```

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;          // 1.0`
  - `double y = 1 + (double) 1 / 2;          // 1.5`

- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`

- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`

Examples (Type Casting)

(int)4.8	has value	**4**
(double)5	has value	**5.0**
(double)(7/4)	has value	**1.0**
(double)7 / (float)4	has value	**1.75**