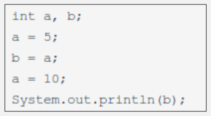


```
int a, b;
a = 5;
b = a;
a = 10;
System.out.println(b);
```

statements are instructions carried out at that moment, not statements of fact that persist into the future

it is better to read assignment statements as “a gets the value 5” rather than “a equals 5”

Arithmetic Operators

- arithmetic operators: +, -, *, /, %
  - / works differently for `int` and `double`
    - if either *a* or *b* are doubles, `a/b` is what you would expect
    - if both *a* and *b* are integers, `a/b` does *integer division* and is the number of whole times *b* divides *a*
  - `%` is mod (modulus) – `a % b` is the remainder when *a* is divided by *b*
    - typically only applied to `int`, but it does work for `double` as well

`8.5/2.5` is 3.4	`8/2` is 4	`7 % 5` is 2
`2.5/12.5` is 0.2	`15/2` is 7	`23 % 5` is 3
`7.5/2.5` is 3.0	`8/10` is 0	`5 % 13` is 5
`8.0/10.0` is 0.4		
`8.0/10` is 0.4		
`8/10.0` is 0.4		

- string concatenation: +

`“hi”+” “+”there”` is “hi there”
`“gr”+8` is “gr8”

Type Conversion

- type conversion occurs when a value of one type is used in a context where a value of another type is needed
- *automatic type conversion* occurs in cases when the conversion can be done without changing the semantics of the value
  - e.g. `double x = 2;` is legal – even though 2 is an `int`, automatic type conversion takes place because 2.0 and 2 represent the same quantity and 2.0 is stored in `x`
- *type casting* is required otherwise – the programmer must explicitly OK the loss of precision
  - e.g. `int x = 2.7;` is not legal – an `int` can’t store a value with a decimal point, and dropping the .7 changes the value
    - solution is `int x = (int)2.7;` – x gets the value 2 (the decimal is dropped)
  - the types must still be convertible – cannot, for example, cast a `String` to `int`

Type Conversion in Expressions

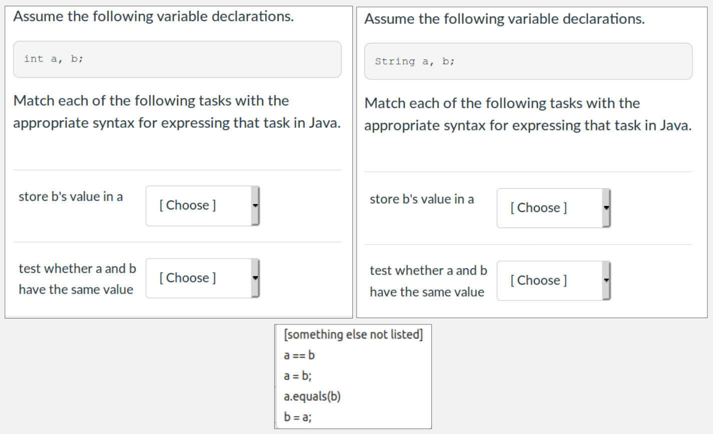
- arithmetic operators expect both operands to be of the same type
  - if both are `int`, the result is `int`
  - if at least one is a `double`, any `int` values are automatically converted to `double` and the result is `double`
  - for +, if at least one operand is a `String`, the other is converted to a `String`

8+2 is 10
`8.5+2` is 10.5
`4+8.5` is 12.5
`5.2+3.8` is 9.0

```
int a, b;
a = 10;
b = 4;

System.out.println(a/b);            // 2
System.out.println(1.0*a/b);        // 2.5

System.out.println(5.5+a/b);        // 7.5
System.out.println(5.5+1.0*a/b);    // 8.0
```



Assume the following variable declarations.

```
int a, b;
```

Match each of the following tasks with the appropriate syntax for expressing that task in Java.

store b's value in a [Choose]

test whether a and b have the same value [Choose]

Assume the following variable declarations.

```
String a, b;
```

Match each of the following tasks with the appropriate syntax for expressing that task in Java.

store b's value in a [Choose]

test whether a and b have the same value [Choose]

[something else not listed]
a == b
a = b;
a.equals(b)
b = a;

Testing for Equality

Guidelines –

- use == for primitive types
  - *primitive types* are lowercase types like `int`, `double`, `boolean`, `char`
- use `s1.equals(s2)` for `String`
  - `s1` and `s2` are `String` variables or literals

Developing Programs

- as soon as you identify the programming construct to use, write down its template
  - write a program → program structure template
  - store a value → variable declaration and initialization
  - read input → set up `Scanner`
- the statements in `main` are executed one at a time, starting with the first one
  - identify the first step, the second step, the third step, etc
  - write pseudocode in comments
- incremental development – write small amounts of code and test frequently
  - write code for one step at a time
  - start with a simpler version of a task
    - e.g. initialize a variable with a hardcoded value instead of reading user input
  - use `System.out.println` to check values of variables

Input Using Scanner

– *input* refers to obtaining values from outside the program e.g. *user input* is something the user of the program provides

- tell the system where to find the definition of `Scanner`
  `import java.util.Scanner;`
  - put at the very beginning of the program file, before the `public class` ...
- create a `Scanner` to read input typed in on the keyboard
  `Scanner` *`stdin`* `= new Scanner(System.in);`
  - this is actually a variable declaration and an assignment statement combined into one
  - *stdin* can be any variable name you want
    - "stdin" stands for "standard input", which is a common name for input from the keyboard
  - this goes inside `main`, before the "read input" step(s)
- read input
  `stdin``.nextInt();`
  - this uses the `Scanner` *stdin* to read in the next thing in the input as an integer
    - use `nextDouble()`, `nextBoolean()`, `nextLine()`, ... to read other kinds of values
  - a typical pattern is print a prompt to tell the user what to enter and then to store what is read in a variable (and then use the variable)
    `System.out.println("`*`prompt`*`");`
    `varname` `=` *`stdin`*`.nextInt();`

Types in Java

- primitive types – `int`, `double`, `boolean`, `char`, `short`, `long`, `float`, `byte`
  - built-in types that are part of the language definition
- types defined by classes
  - `String`
  - `Scanner`
  - and many others...
- enums

Enums

- *enumerated types* are types defined by a set of values

```
enum typename { VALUE1, VALUE2, VALUE3, … }
```

  - the definition can go into the same class that uses it (but outside `main` or any other subroutine) or in a separate file *`typename`*`.java`
  - convention is all caps for the values
  - *`typename`* can be used in variable declarations
  - values are referred to with *`typename.VALUE`*
  - type conversions
    - `enum` → `String`: use `enum` value in a `String` context (automatic type conversion)
    - `String` → `enum`: *`typename`*`.valueOf(`*`str`*`)`