system.out ⇒ system output which can be shown on monitor

Date
Page

#Printing

system.out.println()
system.out.print()
↑
showcase data on monitor

println will print data & move cursor to next line
print will print data on screen & ~~move cursor to~~ do not move cursor

⇒ Print functions are there & it has many type In Java there exist overloaded function which are created to detect the type of data inputed & will fetch data accordingly

Compiler will fetch datatype & print data accordingly

⇒ Println will print any kind of data on monitor

You can print you can add any kind of data with string

⇒ Javac is designed such a way that if you're adding ~~op~~ two operands & if one has datatype of string so there will be string Concatanation will take place

You can perform multiple kind of addition in same code

S.O.P(~~"Hello~~ 10+20+86.5~~8~~+"Helo")

Output?

letsee 8 ~~88~~ 8 118.5 Hello

\n will push cursor on next line

Date Page

⊛ Printf & format()

Both are S.O functions

System.out.printf();
system.out.format();

⊛ Printf

System.out.printf(String, arguments).

In string we can specify format specifier & as argument we can fetch value for that reserved format just like printf in C

%d ⇒ format specifier for int
%f ⇒ —— " —— float
%c ⇒ —— " —— for character

Ex int a=10; char c="A"; float f=10.5f;

Printf("Value are: %d, %f, %C", a, f, c);

so ⇒ Values are: 10 10.5 A
output

↳ so such like how you can pass value

you can able to do operation in argument,

int a=10, int b=120;
Printf(" a + b: %d", (a+b));

a + b: 130

\n will
~~Push text to~~

Road

%number$

0.0012 ⇒ scientific rep
12×10^{-4}
1.2×10^{-3} 1.2e-4

you only can write character in single quotes
char a='a'; ✓ Char a="a"; X wrong

If you want to manipulate number of argument

int x=10; char xx='a'; float z=10.5f;
~~int y~~
%number$ Access specifier
Printf("%3$f %1$d %2$c", x,xx,z);

so even we passing float last we can use it firstly

⇒ you can use single argument many time
int x=10,
Printf("%1$d %1$d %1$d", x);

%1$d
↑ I access specifier
Indexflag which point to index at which we want

(note) ⇒ Count of argument will start from 1 not 0

⊕ formating of values in Java

case 1 — space flag ⇒ width ⇒ 5, flag ⇒ ' ' (none)

%5d, 10 — output ---10

allocate 5 place & print 10 in that

(empty space)

case 2 — If you write (%05d, 10) — width ⇒ 5, flag ⇒ 0

so now it will occupy 5 place & empty places will take 0 value

Output ⇒ 00010

⇒ 0 will fill empty places with 0 (zero)

case 3 — If you write (%(5d, -10) — width ⇒ 5, flag ⇒ (

so if no. is - negative it will show in () & as show width will be same

_(10) ⇒ space

(%(5d, +2) ⇒ output ----2

(4) If you write (%+5d, -10) — width ⇒ 5, flag = +

+ flag used to show signed values

+ will show sign positive if argument is + & - negative if argument is negative

output ⇒ --10

(%+5d, 10) ⇒ --+10

#format specifier summary

%o [argument_index $] [flags] [width] [.precision] [conversion]

↑ Access specifier

%02$

flg flags ⇒ '-', '+', '0', ' ', '('

argument_index ⇒ 1$, 2$, 3$ ....

so

flg

[%0@-5d, -10] ⇒ ---10

(%0-5d, 10) ⇒ ---10

show sign if it is negative

⇒ Ok so now move to precision

for ~~prestion~~ Precision we will need the float element

* float a = 123.45f;

so we write precision in 2 places after per & you have to gave some width

& flags work same for float as they work for int

:n => number after the decimal point

width => 6 Precision 2

```
System.out.printf("%6.2f", a);
```

a — 6 ,

=> Precise

this formatting is very powerfull & you can optimize your code.

If you wright width + it will give blanks from left side

%20f

| _ _ _ Java

% -20f

Java _ _ _ _ |

Such like that

And

```
System.out.printf("%20s", str);
```

is same like

```
System.out.format("%20s", str);
```

Both works same