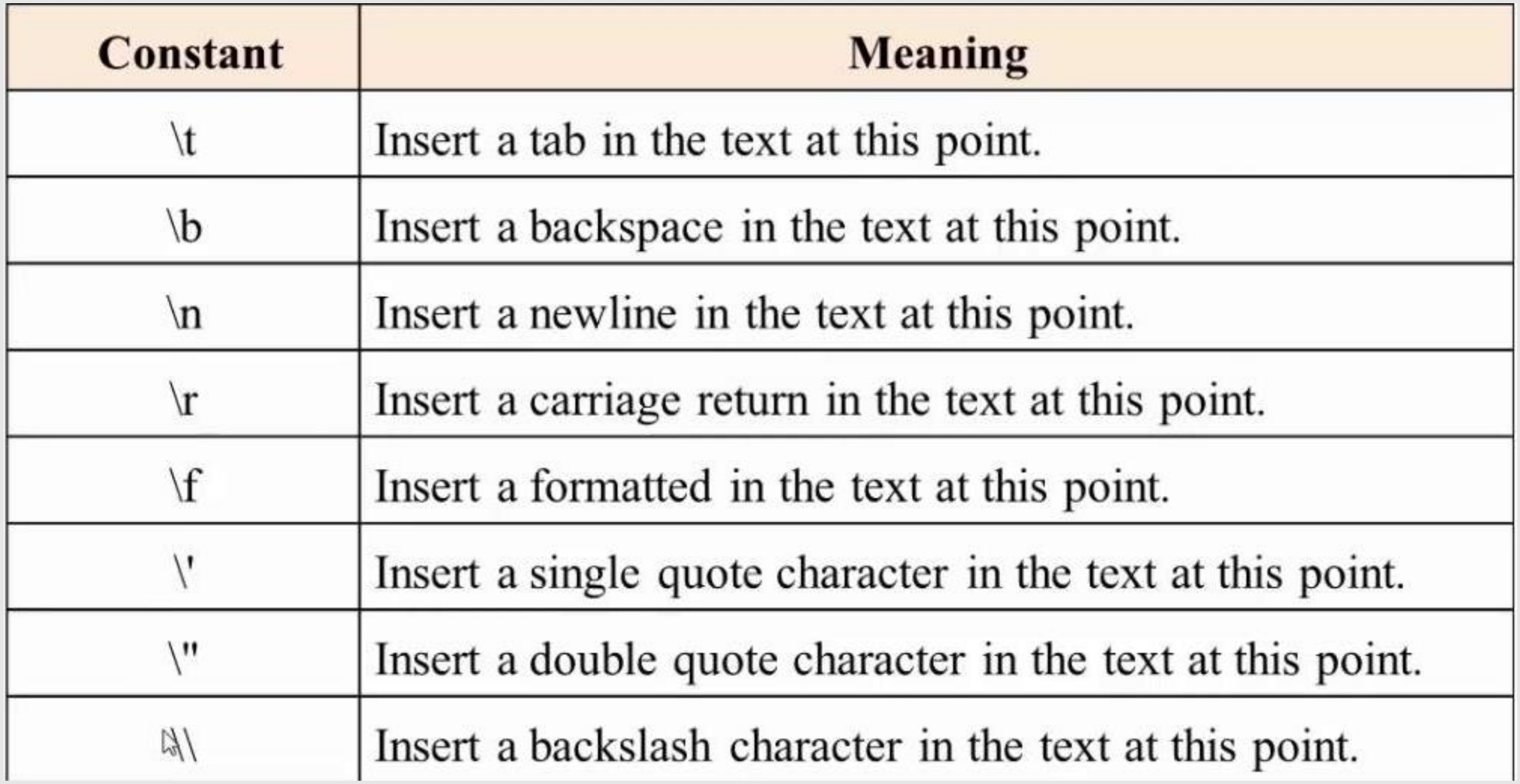
JAVA MIDTERM (PART 2)

Siwat Sirichai

What is ‘\’ (Backslash) in SYSOUT

- \ is used with an escape character
- \ can be used to print illegal character (Illegal character includes ’,”,\)
- Can be use to manipulate the output cursor

Escaping Character in SYSOUT (FAST)



Constant	Meaning
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formatted in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Explanation of complex escape character

- \r | move the cursor to the first character of the line.

Example: “ABCDEFGHIJKL\rMNOP” prints “MNOPEFGHIJKL”

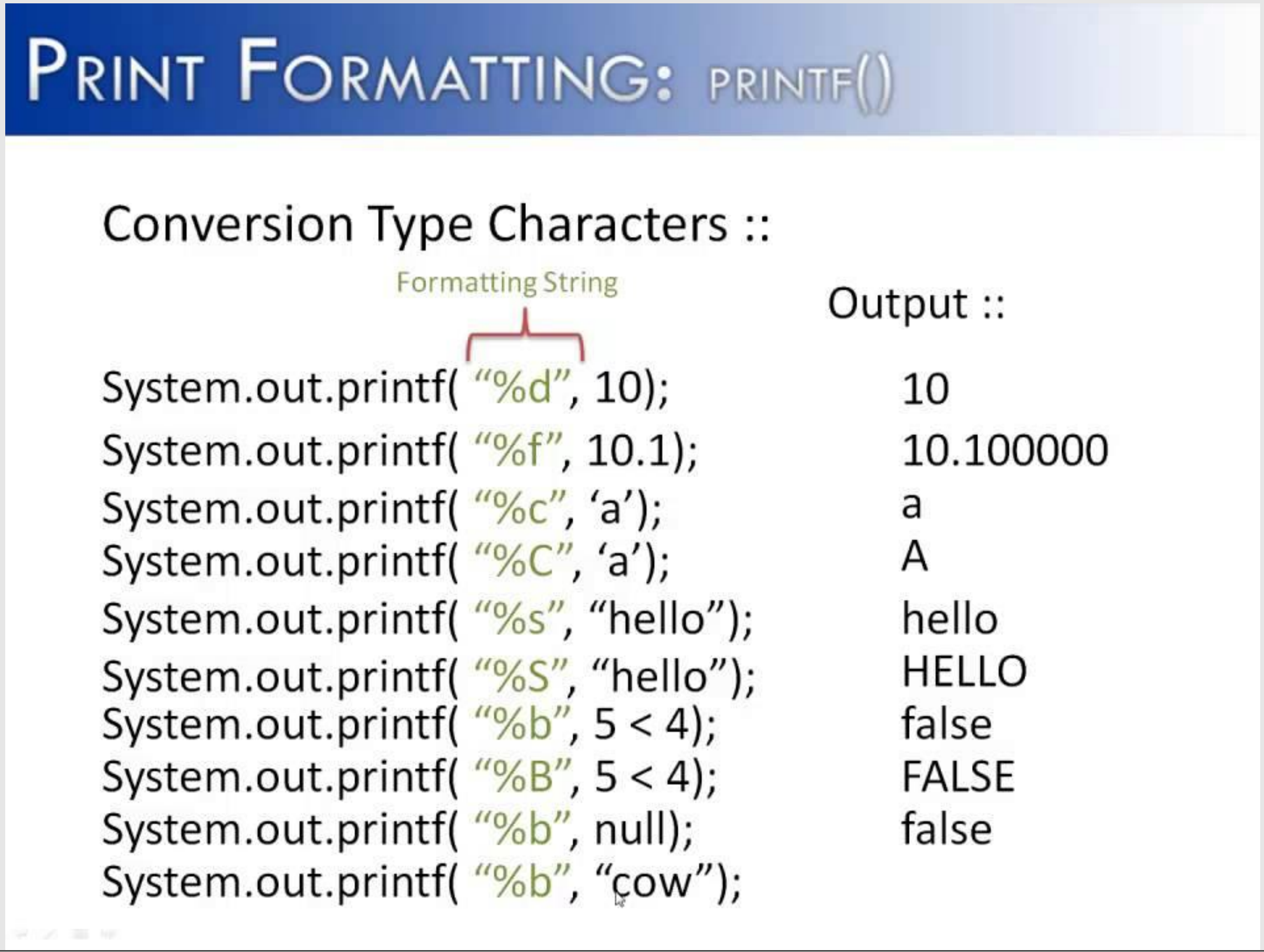
- \f | indicate a line feed, (Used in type printer)
- \b | delete the character before the cursor

Formatting sysout

◦System.out.printf(“%d”,123); //Print 123

“%fix”,datatype

System.out.printf("%",data);

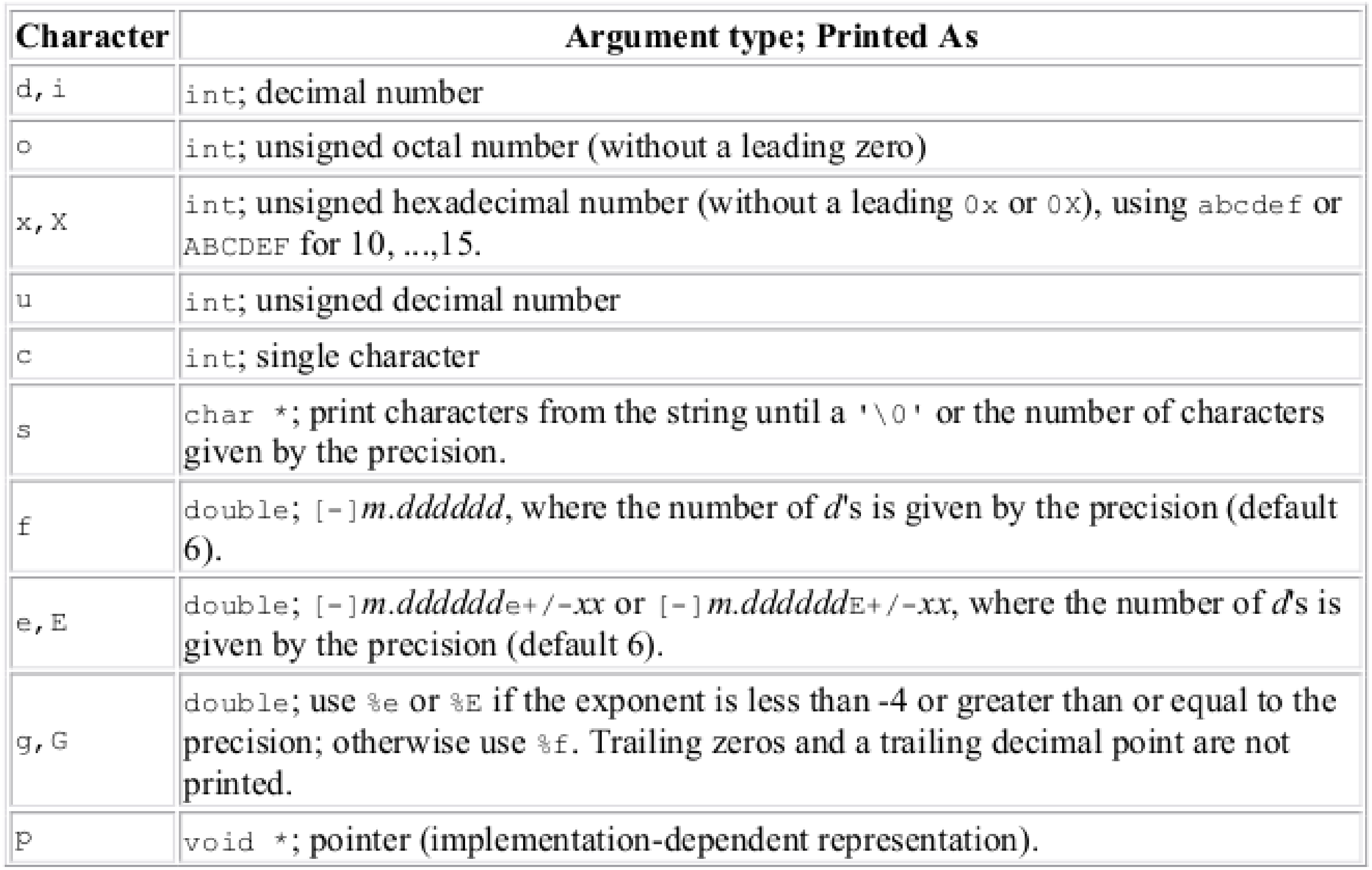


PRINT FORMATTING: PRINTF()

Conversion Type Characters ::

```
System.out.printf( "%d", 10);              10
System.out.printf( "%f", 10.1);            10.100000
System.out.printf( "%c", 'a');             a
System.out.printf( "%C", 'a');             A
System.out.printf( "%s", "hello");         hello
System.out.printf( "%S", "hello");         HELLO
System.out.printf( "%b", 5 < 4);           false
System.out.printf( "%B", 5 < 4);           FALSE
System.out.printf( "%b", null);            false
System.out.printf( "%b", "cow");
```

Output ::



Character	Argument type; Printed As
`d, i`	`int`; decimal number
`o`	`int`; unsigned octal number (without a leading zero)
`x, X`	`int`; unsigned hexadecimal number (without a leading `0x` or `0X`), using `abcdef` or `ABCDEF` for 10, ...,15.
`u`	`int`; unsigned decimal number
`c`	`int`; single character
`s`	`char *`; print characters from the string until a `'\0'` or the number of characters given by the precision.
`f`	`double`; `[-]`*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
`e, E`	`double`; `[-]`*m.dddddd*`e+/-`*xx* or `[-]`*m.dddddd*`E+/-`*xx*, where the number of *d*'s is given by the precision (default 6).
`g, G`	`double`; use `%e` or `%E` if the exponent is less than -4 or greater than or equal to the precision; otherwise use `%f`. Trailing zeros and a trailing decimal point are not printed.
`p`	`void *`; pointer (implementation-dependent representation).

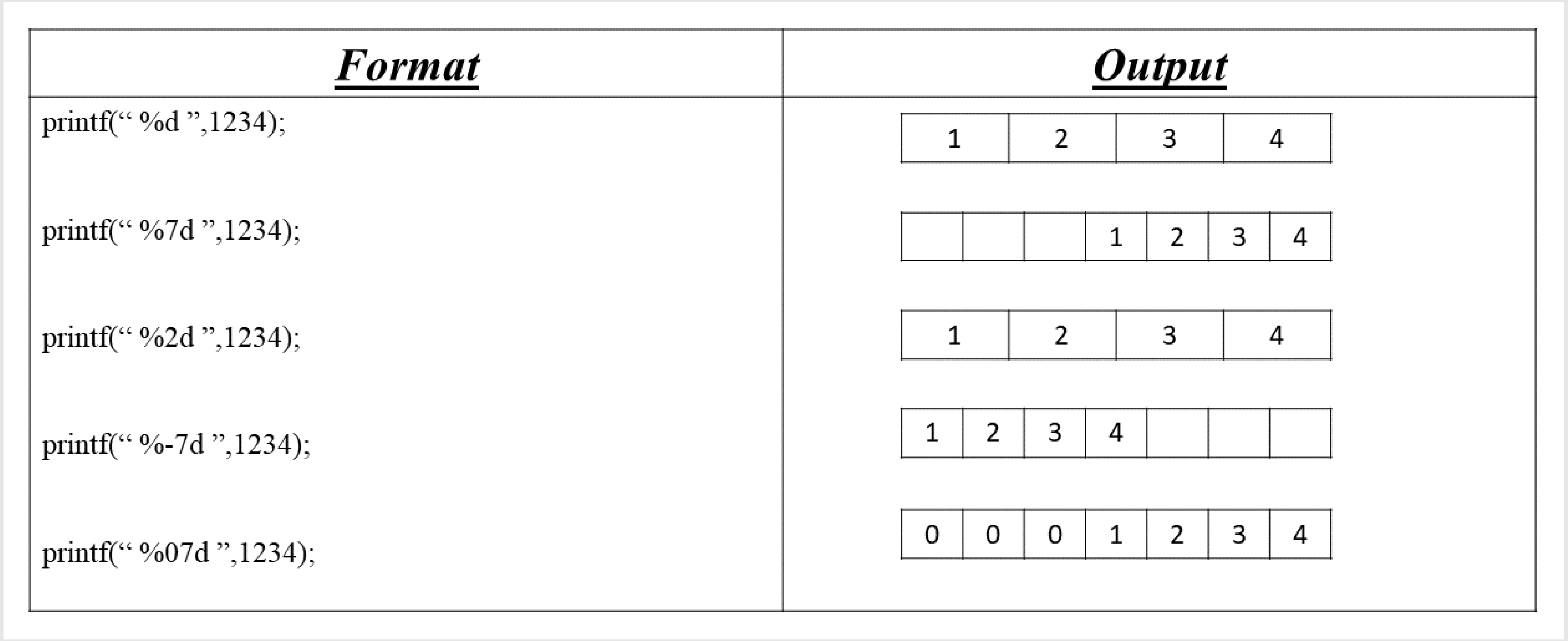
Formatting Syntax Table

You will need to be able to remember this.

If you can't just remember all of it just remember

1. %d with int
2. %f with double
3. %s with String

Number formatting



Format	***Output***						
printf(" %d ",1234);	1	2	3	4			
printf(" %7d ",1234);				1	2	3	4
printf(" %2d ",1234);	1	2	3	4			
printf(" %-7d ",1234);	1	2	3	4			
printf(" %07d ",1234);	0	0	0	1	2	3	4

1-4 Applies with String too!

String formatting Syntax

- %xs means the output occupy at least x character, the output is at the furthest right with blank space on the left filled with ‘ ‘ [Space]
- %-xs means the output occupy at least x character, the output is at the furthest left with blank space on the right filled with ‘ ‘ [Space]

Number formatting Syntax #2

- %-x[d/f] means that the output occupy at least x character, the output is at the furthest left with blank space on the right filled with ' ' [Space] for d and '0' for f

"%-10f",1234.5678 = "1234.56780" | "%-3d",12 = "12 "

- %-x.yf means the output occupy at least x character (including the '.' [dot]) and the decimal occupy exactly y character, the output is at the furthest right with blank space on the left filled with ' ' [Space]

"%10.2",1234.5678 = "   1234.57" | "%10.2f",1234.5678 = "1234.57   "

Number formatting Syntax

- %x[d/f] means that the output occupy at least x character, the output is at the furthest right with blank space on the left filled with ‘ ‘ [Space]

“%3f”,1234.5678 = “1234.5678” [data length > format length] | “%3d”,12 = “ 12” [data length < format length]

- %0x[d/f] means the output occupy at least x character (including the ‘.’ [dot]), the output is at the furthest right with blank space on the left filled with ‘0’

“%010f”,1234.5678 = “01234.5678” | “%010d”,32 = “0000000032”

- %x.yf means the output occupy at least x character (including the ‘.’ [dot]) and the decimal occupy exactly y character, the output is at the furthest right with blank space on the left filled with ‘ ‘ [Space]

“%10.2”,1234.5678 = “   1234.57” | “%3.2f”,1234.5678 = “1234.57”

- %0x.yf means the output occupy at least x character (including the ‘.’ [dot]) and the decimal occupy exactly y character, the output is at the furthest right with blank space on the left filled with ‘0’

“%010.2”,1234.5678 = “0001234.57” | “%03.3f”,1234.5678 = “1234.568”

Rules for Numerical Formatting

- Decimal is rounded up or down according to it’s value
- **%d for int %f for double**
- % can be use with inline string too

Example:

“Hello %f World”,123.23 prints Hello 123.23 World

- Multiple % can be used like so

“%f %f %f”,1.2,3.4,5.6 prints “1.2 3.4 5.6”

Switch Statement Syntax

```
switch(expression){
        case expression1:
                statement1;
                statement2;
                …
                break;
        case expression2:
                statement3;
                break;
        default:
                statement 4:
                break;
}
```

only one case that it’s expression match that of expression in the switch bracket is selected.

Rules Switch Statement

- Only one case in is selected
- default case is not necessary
- can only be used with primitive type (and String) [Quoting JavaDoc: "*A compile-time constant expression is an expression denoting a value of **primitive type** or a **String** that does not complete abruptly."*)
- the final break is not required

Example for Switch

```
int x=1;
switch(x){
        case 1:
                System.out.println(“Hello”);
                break;
        case 2:
                System.out.println(“World!”);
                break;
        default:
                System.out.println(“Goodbye World!”);
}
```

Result: Hello

if-else chain VS switch statement code

```
int x=1;
switch(x){
        case 1:
                do1();
                break;
        case 2:
                do2();
                break;
        default:
                do3();
}
```

is equal to

```
int x=1;
if(x==1)do1();
else if(x==2)do2();
else do3();
```

if-else chain VS switch statement

- switch statement is much faster
- if-else can handle non-primitive type with the .equals method