* Conditionals in Java.

or control statement first part

⊙ Decision making instructions in Java

- If-Else statement
- Switch statement.

> Control statements
> 1> Conditional
> if else, switch, if-else if-else
> 2> Iteration :- Loops

⊙ If-Else statement.

The syntax of If-Else statement in java looks like in C, C++ and JS.

```
if (condition-to-be-checked){
        statement-if-condition-true;
}
else{
        statement-if-condition-false;
}
```

- Example

```
int a =29;
if (a>18){
   System.out.print("You can drive")
```

! Note: Else block is optional.

▸ Relational operators in java.

Relational operators are used to evaluated cond^n (true or false) inside the if statements. Some examples of relational operators are:

==, >=, >, <, <=, !=

↙ equals. → Not equals

! '=' used as assignment operator and '==' is used for comparision / equality check.
! Condition can be either true or false.

* && (Double ampersand), || (pipe operator) and ! (not) are most commonly used operators in Java.
+ They are read as:

&& → Logical AND
|| → Logical OR
! → NOT

} used to provide logic to program.

▸ AND operator
Evaluates to true if both the conditions are true

T && T → T
T && F → F
F && F → F
F && T → F

True → T
False → F

▸ OR operator
Evaluates to true if any one condition is true.

T || T → T
T || F → T
F || F → F
F || T → T

▸ NOT operator
Negates the given input
Like True becomes false and false becomes true.

!T ⇒F
!F ⇒T

- Else-If dause /statement

- Insted of using multiple if statement we can also use else if along with if thus it forms a if-else if-else ladder.
- using such kind of logic reduces indents
- Last else statement will only get executed if all the conditions get fail.

- Syntax

```
if (condition){
    // statement
}
else if (condition){
    //statement
}
else {
    // statement
}
```

- switch case control Instruction.
  Switch case is only used when we have to make a ~~choic~~ choice betⁿ multiple alternatives for a given variable.

+ Syntax:-

```
switch (var){
    case 1:
        // code;
        break;
```

```
case 2:
      //code;
      break;
case 3:
      //code;
      break;
default:
      //code
}
```

- Enhance switch

! No need of break statement.

```
switch (var){
  case 1 -> //statement;
  case 2 -> //statement;
  case 3 {
      // statement
  }
```

! Here in Java we can pass int, char and string arguments for switch.

! Switch can occur within another bt it is rarely used.

Iteration Control Statements

DATE

PAGE NO. 19

(Loops)

- Sometimes we want our program to execute a set of instructions over and over for again & again. For ex. print 1 to 100, etc.

- Loops make it easy for us to tell the computer that a given set of instructions need to be executed repeatedly.

⊙ Types of loops.
Primarily there are three diff type of loops.

1) while loop
2) do while loop
3) for loop.

1) while loop.
Syntax:

```
while (boolean cond^n) {
    // statement
}
```

→ This will be keep executing as long as condition is true.

! If the given cond^n never get false, the while loop keep getting executed. Such a loop is known as infinite loop.

2) Do While loop.
This loop is similar to a while loop except the fact that it is guarranteed to execute at least once.

Syntax:

```
do{
    // code/statement      ← This will executed at least once.
} while(condition);
```

IMP

Note

- While: First condition get checked then the code will get executed.

- do-while :- Here first code will executed then condition will get checked.

- Here code/statement will get executed at least once in the case of conditions are true or false that will not matter for this. but in the case of ~~wit~~ while loop first condition will get checked then it will enter in the loop.

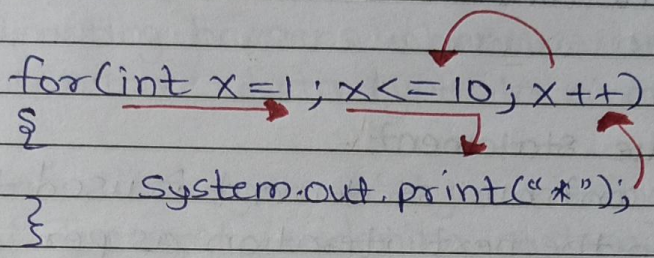
3> for loop.

Syntax

```
for(initialize; Check-bod-expression; update)
{
    // code
}
```

▲ For loop is usually used to execute a piece of code for specific number of times.

Flow of execution



→ Get initialized
→ check cond^{n} if true then execute code, if not treue/false it will break loop.
→ will perform given task like x++, x--, etc.
→ Then again condition will get checked if true then code get executed and if false then ~~code~~ loop get stoped.
→ This will goes on till the cond^{n} get false.

Decreamenting for loop

```
int i = 0;
for(i =7; i!=0; i--){
        System.out.print(i);
}
```

This for loop will run until the value of i become equal to "0"

Break ~~Statement~~ statement.

The break ~~loop~~ is used to exit the loop irrespective of whether of the condition is true or false.

- As when ever 'break' is encountered inside the loop, the control is sent to outside of the loop.

⊙ Continue statement.

The continue statement is used to immediately move to the next iteration as per given cond^n.

In a brif:

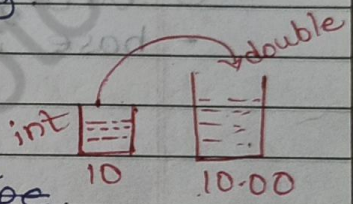
- Break statement completely exits the loop.
- Continue statement skips the iteration/ perticular iteration of the loop.

Type Casting.

Type casting happens when we assign a value of one primitive data type to another type.

There are two types of type casting
1) Implicit Typecast / widening casting
2) Explicit Typecast / narrowing casting

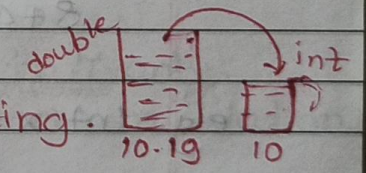
1) Implicit typecast / widening casting.



- Converting smaller size to larger ~~type~~ size.
- It is automatically done by compiler
- It is also called as upcasting / widening.
- no loss of data happens here.

byte → short → char → int → long → float → double

2) Explicit typecast / narrowing casting.



- Converting larger size to smaller size.
- It is manually done by programmer.
- It is also called as downcasting / narrowing.
- Here is chance of data loss.

double → float → long → int → char → short → byte